UNIVERSITY OF CALIFORNIA
SANTA CRUZ

CROSSTON TAVERN: MODULATING AUTONOMOUS CHARACTERS BEHAVIOUR THROUGH PLAYER-NPC CONVERSATION

A thesis submitted in partial satisfaction of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTATIONAL MEDIA

by

Elisabeth Oliver

June 2021

The Thesis of Elisabeth Oliver
is approved:

Professor Michael Mateas, Chair

Professor Adam Smith

Quentin Williams
Interim Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

Abstract

Crosston Tavern: Modulating Autonomous Characters Behaviour through Player-NPC Conversation

by

Elisabeth Oliver

NPCs (Non-Player Characters) are a staple of video games, filling all kinds of supporting roles. This work seeks to flip that paradigm and place the player in the role of support for the goals of a small collection of NPCs. Built on a world simulation where NPCs can take action according to their goals and knowledge of the world state and a conversation space in which the NPC is able to report their actions and exchange information with the player, this prototype AI-based game design explores a new player-NPC interaction in which player conversational actions indirectly influences NPC action. Here we discuss the architecture of these systems, provide a design postmortem, and report some initial player feedback.

To my siblings.

Acknowledgments

This work was done under the guidance of professors Michael Mateas, Noah Wardrip-Fruin, and Adam Smith, to whom I am deeply grateful.

Chapter 1

Introduction

Few video games are complete without a host of non-player characters (NPCs) to accompany the player on their journey through the game's strange world. Some are quirky and loud, brimming with personality, standing front and center of the game, acting as the main cast of their stories, such as Glados and Wheatley of the Portal series [31,32]. Others exist to fill space, like the many nameless townspeople of Pokemon [11]. This kind is closer to scenery than people rather than fulfilling a primary narrative or gameplay purpose. These are the background characters. The extras.

And throughout gaming, there are NPCs whose whole purpose is to provide direction to the player. To offer quests or side quests. NPCs that want things. Generally, these requests come in the form of specific tasks. "Please do this thing for me," they say, where "this thing" is anything from killing particular monsters to collecting objects to saving the world. Rarely is there room to help the NPC in branching ways. Rarely are the NPCs asking for help in accomplishing the task themselves.

But what if NPCs wanted advisors and not goffers? What would a game look like if the NPCs didn't want the player to collect the ten apples for them, but instead just wanted to figure out where they could get those apples? What if the game focused on helping NPCs by providing information and direction?

That is the focus of this work. Crosston Tavern is a prototype AI-based game design centered around helping a small cast of NPCs find love, friendship, and good food. The player takes the role of a tavern keeper with a small group of townies who are their regular customers. Every evening these townies wander in, order food, and share their struggles and dreams with the player. And every evening, it is up to the player to share stories of other townies or provide suggestions as to what they might try next to help their goals. And every day, these townies wander their simulated world trying actions based on what they know and what they have been told with the hopes of fulfilling their dreams.

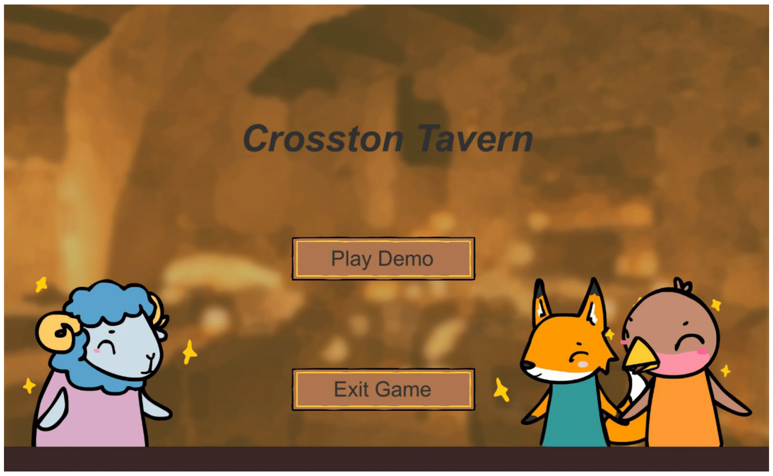


Figure 1.1: The title screen of Crosston Tavern.

Situated in the expressive intelligence [3] research paradigm, this work seeks to flip the standard paradigm of player-NPC interaction. Instead of stationary or scripted NPCs providing support to the player's goals, here we explore giving NPCs goals and a range of actions they can take to achieve them while leaving the player in the role of support, using design patterns common to AI-based game design [7, 30].

Perhaps unsurprisingly, this is not so simple a task. Given this kind of interaction, there is no avoiding that the NPCs are the center point of the game. They have to be the central actors. Their dialogue needs to be complex and rich to reflect their complex decision-making process and their variable actions.

In most games that are NPC focused, this is achieved by writing a lot of scripted dialogue, such as in games like Disco Elysium [33], Hades [12], or Detroit: Become Human [6]. Large pools of potential conversations are written and pulled up as appropriate. However, in those contexts, NPCs are scripted characters. Their goals are predetermined, the contexts in which they interact with each other and the nature of those relationships carefully considered by the authors putting words in their mouths. Each planned variation requires an exponential increase in written content and so in practice how the state of NPCs mutates over the course of play is carefully and artfully constrained.

This would not work for what we wished to accomplish. Rather, we use the results of NPC actions, the state of their goals, and their knowledge of their world to populate what they said to the player. In doing so, we hoped to create dialogue that was dynamic and reflective of the underlying systems.

The setting for this game is based on the tropes common to the Farming Simulation genre (games like Harvest Moon [15]). In games of this type, the player takes the role of a new farmer in a quiet town that has seen better days. The player primarily spends their time improving their farm, whether that is in cultivating their field, raising animals, or upgrading tools and infrastructure. Additionally, they might collect resources from the rich natural world around the town, frequently there are places to fish or to forage. The main goals of these games are generally twofold: improve your farm and become friends (or better than friends) with your neighbors. And always, the key to the hearts of the other townies is through their stomachs.

With this in mind, in designing the setting of Crosston Tavern, many actions were created for the NPCs that a human player would normally have access to in this genre (ex: farming, fishing, and foraging) and NPCs were each given goals related to becoming “better” at their primary profession, just as the player in standard farming sims would want to become a better farmer. Additionally, each NPC was given a romantic crush on another as romancing an NPC is a common feature of the farming sim genre for players. Here it is up to the player to provide the NPCs with enough information and suggestions to help them with these goals.

Or they can nudge them into giving up these initial goals in favor of other dreams. Because, just because these are the goals they begin with, does not mean they are the goals they will keep. Over the course of play, it is possible for their feelings to change and for the NPCs to develop different goals based on those new feelings. So although they begin in a love triangle, that doesn’t mean that one character is doomed to forever hold onto their unrequited love.

Through the game's systems, we create an experience in which NPCs are capable of taking nonscripted action toward their own goals and of reporting that action back to the player in interesting ways. Additionally, we put the player in the role of support of these NPCs, reversing the usual player-NPC interaction dynamic.

In the remainder of this thesis, we first briefly describe play of the created system for context. We then discuss similar games, systems, and academic research that put focus on NPCs, player-NPC interaction, or indirect NPC control. We then go into detail on the previously mentioned architecture of Crosston Tavern, starting with the world simulation, moving into the conversation space, and finishing with the interaction between the two. This is followed by a report on our preliminary playtests of the system, then a description of future work.

1.1 Description of Gameplay

Gameplay begins with one of the three NPCs (Figure 1.2) entering the player's bar. The conversation begins with the NPC greeting the player, perhaps mentioning whether they had a good day or not, and then ordering food. From there, the conversation opens: the player can ask the NPCs a number

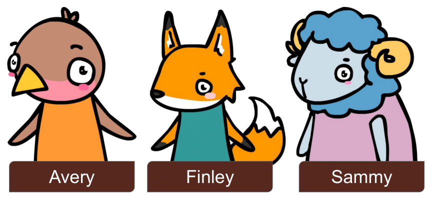
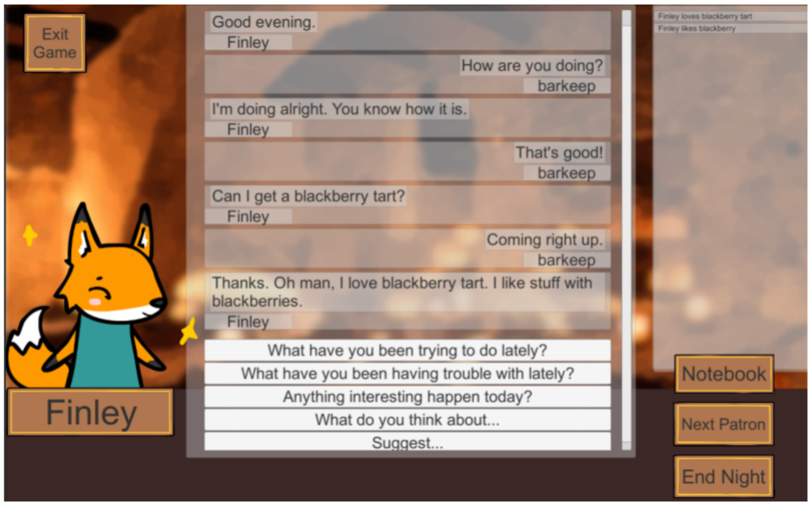


Figure 1.2: The three NPCs of Crosston Tavern: Avery the bird, Finley the fox, and Sammy the Sheep.

of questions, ranging from inquiring what the NPC did that day, to what their goals are, to what they think of other NPCs (Figure 1.3).



Good evening.
Finley

How are you doing?
barkeep

I'm doing alright. You know how it is.
Finley

That's good!
barkeep

Can I get a blackberry tart?
Finley

Coming right up.
barkeep

Thanks. Oh man, I love blackberry tart. I like stuff with blackberries.
Finley

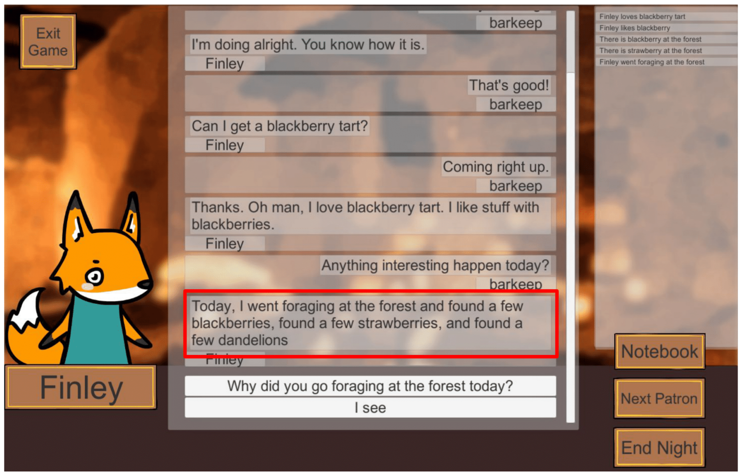
What have you been trying to do lately?
What have you been having trouble with lately?
Anything interesting happen today?
What do you think about...
Suggest...

Finley loves blackberry tart
Finley likes blackberry

Figure 1.3: The player has a variety of things they can say to an NPC. Without any system knowledge (which can unlock additional dialogue options) the player has access to:

- “What have you been trying to do lately?” - What are your current goals?
- “What have you been having trouble with lately?” - What goals are you unable to make any progress toward?
- “Anything interesting happen today?” - What actions did you take today or happened around you that you think are interesting enough to report?
- “What do you think about. . . ” - What do you think of another NPC?
- “Suggest. . . ” - Suggest an action for Finley to do in the simulated world.

The answers NPCs provide to these questions are derived directly from the state of the world simulation that happens before the game starts and between each night at the bar. For example, the player might ask “What did you do today?” This queries the NPC’s knowledge base of remembered events and filters for the ones that happened on the current day. These events are then curated to a short list of "interesting" (from the NPCs point of view at least) events to tell the player. This might look something like Figure 1.4. We will go into more detail on how the NPC decides which events are interesting and how the system representations of those events become English text in more detail later (Section 3.2.3: Verbalization).



barkeep

I'm doing alright. You know how it is.
Finley

That's good!
barkeep

Can I get a blackberry tart?
Finley

Coming right up.
barkeep

Thanks. Oh man, I love blackberry tart. I like stuff with blackberries.
Finley

Anything interesting happen today?
barkeep

Today, I went foraging at the forest and found a few blackberries, found a few strawberries, and found a few dandelions
Finley

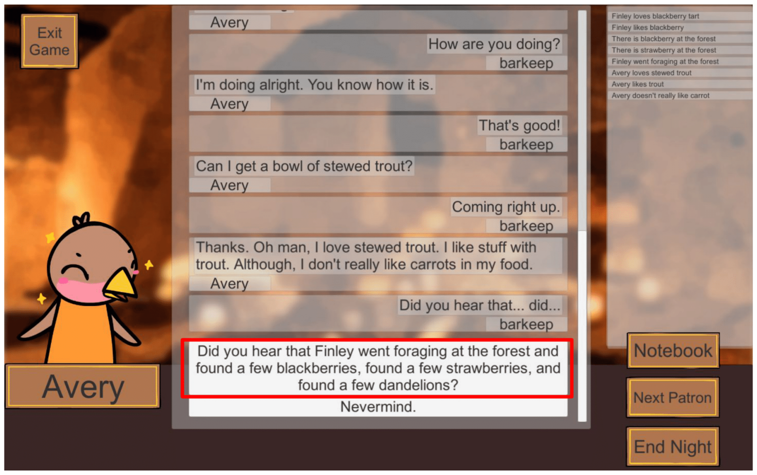
Why did you go foraging at the forest today?

I see

Finley loves blackberry tart
Finley likes blackberry
There is blackberry at the forest
There is strawberry at the forest
Finley went foraging at the forest

Figure 1.4: The dialogue of NPCs is populated by their knowledge of world state. In this case, the NPC Finley is answering the player question "Anything interesting happen today?" with a report on the action they thought was interesting (foraging in the forest). In principle, an NPC can respond to this question with a list of several actions, some of which could have been performed by another NPC.

Alternatively, the player might also share events or other information that they heard from other NPCs. For example, if the above dialogue came from Finley, the player would be able to tell Avery or Sammy about Finley's foraging exploits when either of the others next steps into the bar (Figure 1.4). By doing this, the player can indirectly tell NPCs where they might find certain resources in the world, in this case where to find, strawberries, blackberries, and dandelions (Figure 1.5).



Avery

How are you doing?
barkeep

I'm doing alright. You know how it is.
Avery

That's good!
barkeep

Can I get a bowl of stewed trout?
Avery

Coming right up.
barkeep

Thanks. Oh man, I love stewed trout. I like stuff with trout. Although, I don't really like carrots in my food.
Avery

Did you hear that... did...
barkeep

Did you hear that Finley went foraging at the forest and found a few blackberries, found a few strawberries, and found a few dandelions?

Nevermind.

Finley loves blackberry tart
Finley likes blackberry
There is blackberry at the forest
There is strawberry at the forest
Finley went foraging at the forest
Avery loves stewed trout
Avery likes trout
Avery doesn't really like carrot

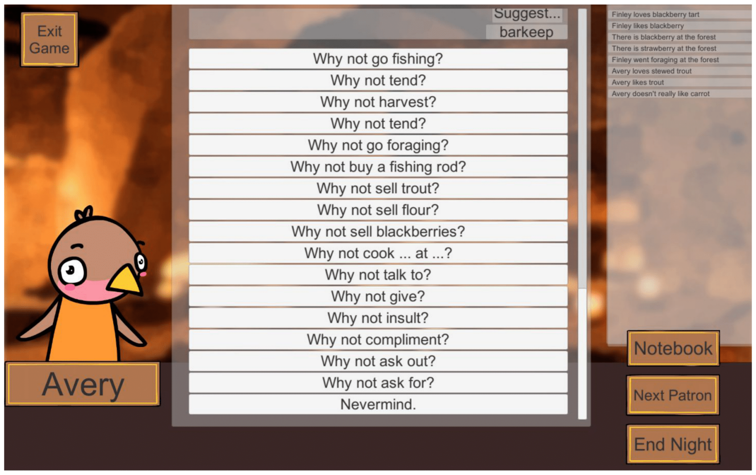
Figure 1.5: The player can tell NPCs about actions other NPCs have told them about. In this case, Finley had told the player about their foraging action earlier and the player is now passing that information along to Avery.

Additionally, the player is able to suggest NPCs take specific actions in the world (Figures 1.6 and 1.7). These actions are registered as high-priority goals for the NPCs to fulfill in the world simulation and can radically alter what they choose to do in a given day. These goals remain until a player tells the NPC to stop. Up to three goals can be specified at a time.

Each evening, each conversation with each NPC lasts anywhere from three to seven “moves” depending on how much they like the food they ordered. Each “move” consists of the player either asking a question and getting an answer from the NPC or telling the NPC something and getting a reaction. Each evening, each NPC comes to the bar in a random order.

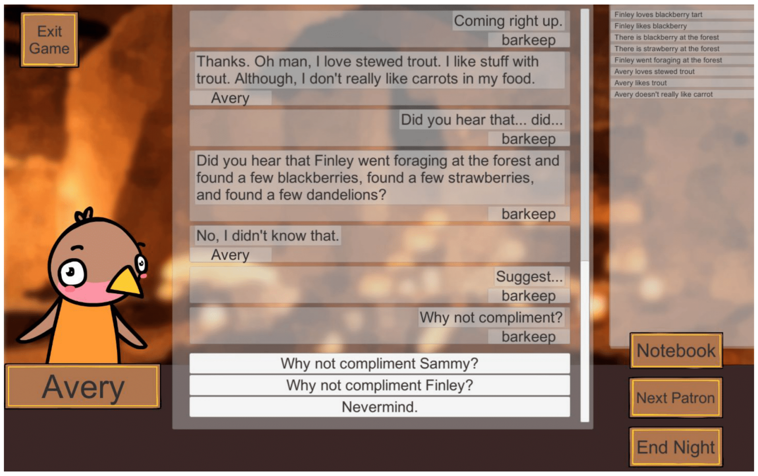
Between each night at the bar (and before play begins), a simulation of the world is run where the NPCs are able to wander about and perform actions in an attempt to satisfy their goals. The details of this simulation will also be discussed in detail shortly.

There is no specific goal that the player is given, but the system is designed around the players wanting to manipulate the relationships between the three NPCs. At the start of the game, the three NPCs are in a love triangle (Figure 1.8). Although their relationships naturally drift over time, the player has a lot of power to pair up the characters as they please or to tear their existing positive relationships apart.



Finley loves blackberry tart
Finley likes blackberry
There is blackberry at the forest
There is strawberry at the forest
Finley went foraging at the forest
Avery loves stewed trout
Avery likes trout
Avery doesn't really like carrot

Figure 1.6: The player may suggest to the NPC they are currently speaking to that they should perform any action available in the system. Each of these options continue to an additional menu where the target of the action can additionally be specified.



Coming right up.
barkeep

Thanks. Oh man, I love stewed trout. I like stuff with trout. Although, I don't really like carrots in my food.
Avery

Did you hear that... did...
barkeep

Did you hear that Finley went foraging at the forest and found a few blackberries, found a few strawberries, and found a few dandelions?
barkeep

No, I didn't know that.
Avery

Suggest...
barkeep

Why not compliment?
barkeep

Finley loves blackberry tart
Finley likes blackberry
There is blackberry at the forest
There is strawberry at the forest
Finley went foraging at the forest
Avery loves stewed trout
Avery likes trout
Avery doesn't really like carrot

Figure 1.7: For example, the compliment action can be used to target either of the other NPCs, Sammy and Finley.

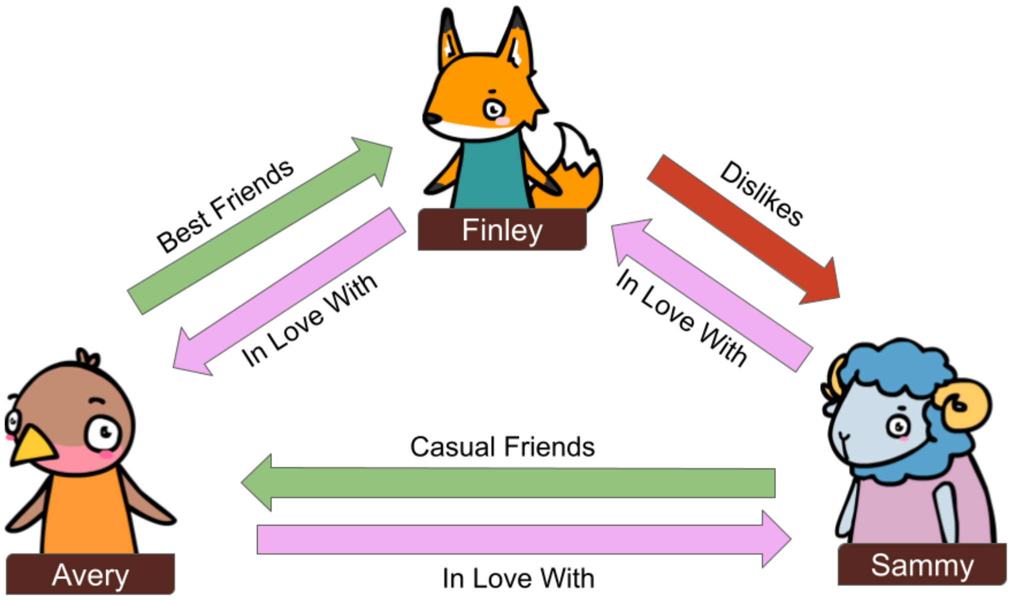


Figure 1.8: On game start, the three NPCs are in a true love triangle: Avery in love with Sammy, who thinks of them as a casual friend; Sammy in love with Finley, who actively dislikes them in return; and Finley in love with Avery, who thinks they are best friends.
Through player intervention, these relationships can change in a wide variety of ways, from all three ending up as close friends, to all three hating each other, to any romantic pairing.

Chapter 2

Related Work

2.1 Farming Sims

Crosston Tavern borrows its setting from the Farming Simulation (Farming Sim) genre of video games (not to be confused with the game series called *Farming Simulator* [26]). This genre is characterized by playing as a newcomer to a small, rural town. Usually, the player character has just inherited the (now heavily dilapidated) family farm from a deceased grandfather [1, 2, 4, 15]. Cleaning up and rebuilding this farm becomes the player's initial goal.

Additionally, the player is encouraged to meet, befriend, and romance the other townspeople. Daily interaction, as well as frequent gifts, are the keys to improving these relationships. These relationships, frequently referred to as "heart level", are usually represented from zero to ten with a series of hearts. In most games of this genre, there are events (frequently accompanied with cut scenes) that trigger at certain heart levels.

These scenes are usually cute encounters between the player and the character that provide further characterization. These are generally considered rewards for improving your relationship. Picking a particular bachelor/bachelorette, romancing them, and starting a family with them is usually another major player goal. Additionally, becoming friends with the other townspeople is a secondary goal with varying levels of emphasis depending on the game.

Finally, there is usually a weak central plot focusing on the relationship between the town and nature. Frequently, the town was once blessed by spirits or gods of nature but have since lost touch with those spirits (harvest sprites in the Harvest Moon series, earth sprites in *Story of Seasons* [2], "runies" in *Rune Factory* [21], "junamos" in *Stardew Valley* [4]). By collecting various resources or growing certain crops this connection is reestablished and the power of the spirits, and by extension the land itself, is revitalized.

Of these goals, the NPCs of Crosston Tavern lean most strongly on the second, having NPCs want to form interpersonal relationships with one another. To a lesser degree, each NPC has a "profession" related goal, which is somewhat analogous to the player's "fixing up the farm" goal.

Across this genre, the player collects resources in a number of ways. First and primarily, this is done by growing crops on their farm and raising animals such as cows and chickens. Additionally, there are usually wild harvestables such as mushrooms, berries, or wildflowers that can be foraged from the open areas of the world. Fishing in rivers, lakes, and oceans are also very common, as is logging in forests and mining in a many floored mine. These gathered resources are either used to improve farming facilities (such as building or upgrading barns and coops), to make gifts for the townspeople or spirits, or as a gift in and of itself.

These social and resource mechanics became the inspiration for the actions available to NPCs in Crosston Tavern's world simulation. Characters in Crosston Tavern have access to such actions as talking to each other and giving one another gifts (just as the player can talk to and give gifts to NPCs in farming sims), tending their crops, fishing in rivers, and foraging for wild harvestables.

Additionally, the towns in these games have a few standard buildings: a mayor's house or town hall, a general store, a clinic, and an inn or restaurant. It is this last one, the inn or restaurant, that Crosston Tavern uses as its setting for gameplay.

2.2 Simulated Action

Many video games are built on top of agent-based simulation systems of some sort. This is especially common in management games, such as *Prison Architect* [8] or *Planet Zoo* [5], where the thing being managed (prisoners or animals and zoo guests respectively) have needs that the player-manager provides for.

Such a simulation is also the basis for the games of the *Sims* franchise [17], in which play revolves around directing a set of semi-autonomous characters (called sims) in a simulated world. Sims are mainly concerned with fulfilling six basic needs:

hunger, bladder, sleep, social interaction, fun, and hygiene. The AI attached to the sim agents is not terribly good at fulfilling these needs though, and without player intervention, it is not uncommon for sims to starve themselves to death. Additionally, sims have relationships with each other on two axes, platonic and romantic, which can increase or decrease independently. Because it is built on a simulation, the space of playthroughs available to the player is very large. Players are free to tell themselves any story involving suburban life, from finding true love, to mastering skills or careers, to raising a family, to locking strangers in basements for their own amusement.

Crosston Tavern is built on a similar simulation. Like the sims, characters form relationships with each other over platonic and romantic axes and work to fulfill certain personal goals. However, where the sims' goals are in filling their basic needs, the goals of the Crosston townies are to form relationships with others, improve skills related to their professions, and eat food they love.

Tale-Spin [20] is another system (in this case a story generator rather than a video game) built around simulation. It was built as an experiment in automated storytelling and outputted Aesop's Fables-style short stories from the actions and reasoning of characters within its simulation. Like Crosston Tavern, characters within the *Tale-Spin* simulations have goals and are able to reason about the state of the world and other characters using a means-end problem solving framework. However, where the focus of Tale-Spin was to autonomously generate stories based on character problem solving using an early version of hierarchical task network planning, Crosston Tavern generates the day-to-day actions that a cast of characters might talk to the player about using operator planning.

Elsinore [14] is another game that tells stories on top of a simulation system. Specifically, it is a Shakespearean tragedy simulator that tells variations on *Hamlet*, told from the perspective of Ophelia, depending on how the player controls Ophelia. In this game, Ophelia (the player character) is stuck in a time loop, repeatedly attempting to stop the tragedies of the play from occurring. Every time she dies she wakes up again at the beginning of the loop, four days before the end of the play.

In *Elsinore*, the player can observe interactions between other characters to learn new information or interact with the other characters in an attempt to alter the future Ophelia knows is coming. Each of the other characters has their own plans and is able to reason over who knows what and the state of the world. Like in Crosston Tavern, collecting information from other characters and sharing it strategically to incite action in others is a big part of gameplay.

Prom Week [18] is another game built around simulated actions, this time focusing almost exclusively on those in the social sphere. Set in the week before prom in an average American high school, players help direct a cast of high schoolers through a series of social puzzles to get them ready for prom. At any time, players can choose an action for one of the characters in the level's scenario. Which actions are available to a given character is based on what that character "wants" to do according to the simulation system (*Comme il Faut*, or CiF)'s "influence rules" [19]. CiF and its successor system the *Ensemble Engine* [24] use these influence rules to direct NPC action and determine whether those actions are accepted by the other party. This bears a passing resemblance to the goals of NPCs in Crosston Tavern, with goals, like influence rules, being the main input in determining the desirability of a given action. However, this is only a surface-level similarity at best, as CiF is a rule-based utility model for action selection, and its influence-rules act more closely as a means of determining what goals their agents hold, only then recursively determining what actions move toward those goals.

Versu [10] is another storytelling system based around autonomous agents. Like CiF, it is built around a large set of social rules ("social practices"). These social practices are similar to the features of Crosston Tavern's world simulation in that active social practices provide possible actions to the NPCs. Like the NPCs of Crosston Tavern, the desirability of these provided actions is evaluated against the acting NPC's goals.

2.3 Player-NPC Interactions

Social interaction between the player and NPC takes a number of forms across video games. Many provide a one-size-fits-all "interact" action that prompts short quips or musings from NPCs. Some intermix longer cut scenes as well, frequently as rewards for progressing plot or improving character relations. Others provide detailed dialogue trees, with every tree a unique conversation, frequently connected to the plot or the character's personal arc. Others still provide a collection of social interactions to be used at any time.

Animal Crossing [9] is an example of the first, one-size-fits-all interaction. In games of the Animal Crossing series, one plays as a new villager in a small town who is tricked or sweet-talked into buying a home from an entrepreneurial raccoon. The player is encouraged to interact with the other NPCs (called villagers) in the town as they work to pay off the debt from purchasing their new home. These interactions consist of the player walking up to other villagers and "talking" to them, at which point the villagers rattle off one of hundreds of potential dialogue lines, based primarily on which of eight personality types they are.

These dialog lines take one of a couple of forms. Most frequently, it is a humorous quip about life. These quips are unrelated to any game state and are pulled from one of eight pools of dialogue lines depending on their personality. Other times, they might ask the player a question (for example, "What kind of comics do you read?"); the player is then given a small range of answers ("Fantasy", "Super Hero", etc) to choose from. However, beyond their immediate response to the player's answer, these questions do not affect the game state. Lastly, they occasionally reference another villager. In these mentions, they usually talk about their opinion of the other villager, but their opinions seem to have more to do with the personalities of the two villagers than any simulated state. It is surprising that a game in which socializing with one's neighbors is such a big part of gameplay manages to have such a simple social model, but it makes up for the simplicity with extremely large pools of very charming dialogue.

Farming sims, such as *Stardew Valley* [4], take this simple interaction and intermix cut scene events into the gameplay. A cut scene reward is given every two or so heart levels the player raises with a given character. Many of these are events with no input from players but a few have choice points, some of which improve the player's relationship with the character. These events are unique and will not happen a second time in a given playthrough and frequently involve actions not available during standard play (ex: a trip to a sports game, a ride on a character's motorcycle, a boat ride).

In addition to these events, as the player improves their relationship with an NPC the pool of interaction dialogue changes, picking more and more personal dialogue as the relationship develops. For example, one character might say, "Hey. Your name is [Player], right?" at one heart but at six might instead say, "I like your new boots! Your hair looks nice today. Do you want to hang out for a while?".

Overall, the standard interactions are on the boring side, mainly serving as filler between the juicier events. *Stardew Valley*, in particular, suffers from repetitive dialogue; the pools of dialogue for daily interactions are not that large or varied (there are only around fifteen possible lines of dialogue per season, several of which carry over between seasons, which–given a season is 28 days–is not very many). This lack of content is mitigated slightly by limiting the number of times a player can interact with a given character to once per day.

Other games include branching dialogue interactions with characters. This is particularly common in the role-playing game (RPG) genre, with games like *The Elder Scrolls V: Skyrim* [27], *Fallout 4* [28], and *Disco Elysium* [33], as well as narrative-heavy games such as *The Walking Dead Game* [13] or *Detroit: Become Human* [6]. Done well, it allows the player to talk to the characters about deep and nuanced topics or for the NPCs to share heartbreaking backstory or uplifting dreams of the future. Done poorly, it becomes a menu for the player to click through to understand what the plot is supposed to be. In this system, every conversation, and every path through every conversation needs to be written by an author. Additionally, player interactions don't mean anything to the system directly. Player choices might be tied to specific flags or trigger relationship changes, but the dialogue option the player chooses in and of itself means nothing to the systems responsible for executing them.

Across all of these, the player is either unable to affect the nature of their interaction in any meaningful way (as in the case of the one-size-fits-all interaction) or only has access to a few choices in heavily scripted conversations (in the case of branching dialogue). Additionally, what characters talk about is determined entirely by authors at design time with limited room for variation.

Crosston Tavern, on the other hand, leaves the content of NPC dialogue up to NPCs at playtime. They are able to talk about any action they may have taken or witnessed as well as any of their goals and preferences. The player is able to direct conversation around these topics as they please.

2.4 Knowledge Modeling

Talk of the Town [22, 23] is a system that focuses on simulating knowledge propagation and mutation. NPCs in their simulated world observe physical and social traits about others around them and share these observations with one another. As they propagate this information, their observations can mutate, either unintentionally as characters misremember or intentionally as characters lie.

Like James Ryan's system, the NPCs of Crosston Tavern have a model for learning new information about their world. However, in comparison to the complex models of belief and information propagation present in *Talk of the Town*, the knowledge representation in Crosston Tavern takes a simpler approach. In Talk of the Town, NPCs have a confidence in their beliefs as well as recorded reasons as to why they believe what they do. As they get more evidence of a given piece of world state, that confidence goes up. In contrast, the NPCs in Crosston Tavern simply "know" things to be one way or another (whether this is true or not). If they are told something that contradicts what they already know, the old knowledge is overwritten.

For example, if one were to apply Crosston Tavern setting and world state to the *Talk of the Town* system, if a Talk-of-the-Town-Sammy initially believed Talk-of-the-Town-Finley hated them with full confidence, being told by the player that Finley liked them would only weaken that previously held belief and would add a weakly held belief that Finley liked them. In contrast, if Sammy of Crosston Tavern was told that Finley liked them, they would believe it, completely forgetting that they once thought Finley hated them.

Facts of interest to *Talk of the Town* NPCs include things such as another's appearance, occupation, home, and whereabouts. Of these, NPCs of Crosston Tavern might only be interested in the whereabouts (in that, movement actions are recorded in the memories of Crosston townies). These are fairly sensible choices given the domain space of *Talk of the Town* was a murder mystery experience designed for *Bad News* [25].

In such a domain, the whereabouts of characters and their physical traits is of much greater importance. However, these are less interesting in the context of small-town gossip. Instead, Crosston townies are more interested in actions they and others perform, the relationships between themselves and others, and the item preferences of other townies. The ability of *Talk of the Town* NPCs to lie and misremember is of great interest but fell outside the scope of Crosston Tavern.

The CiF [19] system also featured a model of certain kinds of knowledge. First, like Crosston Tavern, characters in CiF have a record of past events. Additionally, it features a "Cultural Knowledge Base" (CKB) which encodes both the opinions each character has for "props" (items such as zombie movies and roses) and the general opinion the world holds for that item. For example, character A might dislike roses, while the world at large thinks they are romantic. This CKB is loosely analogous to the item preferences found in Crosston Tavern.

However, the facts of the CKB in CiF are known to all characters at all times. This differs from the item preferences of Crosston Tavern, in that item preferences are a learnable piece of information. By default, a character in Crosston Tavern only knows their own preferences. Part of gameplay is telling other characters these preferences as the player learns them themselves. Additionally, there is no global consensus about items as there is in CiF.

2.5 Reversing Roles

Crosston Tavern is not the first game to reverse the role of player and NPC. In *Moonlighter* [29], you play as a shopkeeper during the day. You sell all sorts of odds and ends to visiting shop patrons. It is up to you to price your wears such that you make a maximal profit. Too cheap and you don't make any money off them, too expensive and they don't sell. At night, you "moonlight" as an adventurer, diving ever deeper into mysterious dungeons for loot to sell in your shop during the day. This game intentionally makes the player the shopkeeper, a classic role of NPCs across the Role Playing Game (RPG) genre that the game world is modeled after. In the same way, Crosston Tavern seeks to put the player in the role of the barkeeper/tavern keeper/innkeeper that is so commonly a staple of the Farming Sim genre.

Chapter 3

System Architecture

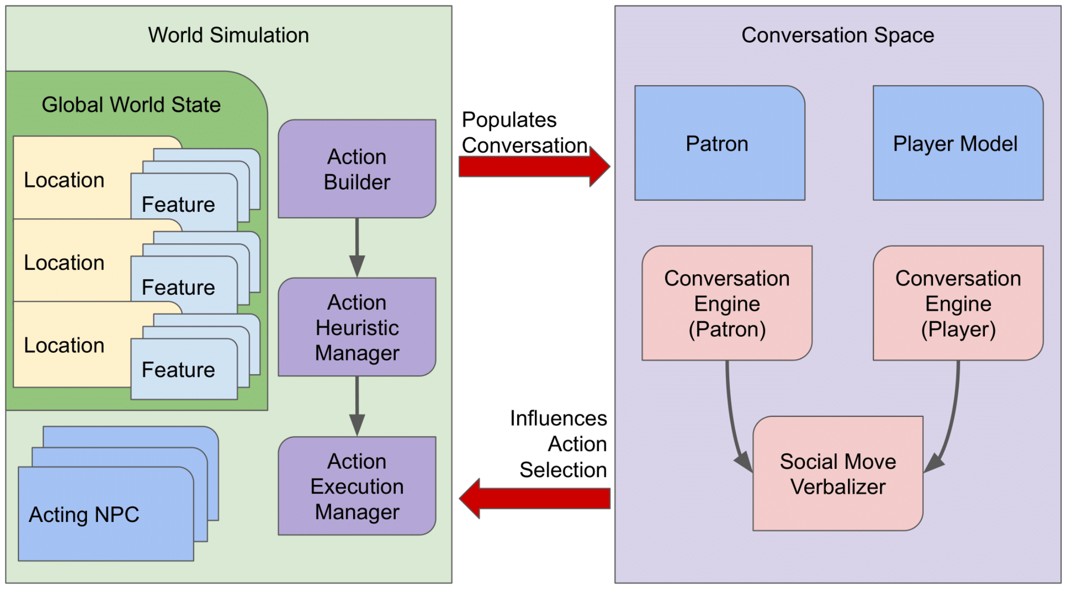


Figure 3.1: Crosston Tavern's systems can be split into two: the Word Simulation and the Conversation Space. World simulation's main purpose is to generate events and world state to discuss during conversation, while actions taken by the player in the conversation space influences the actions taken by NPCs in world simulation. We will be discussing the details further in further sections.

The systems making up the demo can generally be split into two parts (Figure

3.1): the *world simulation* and the *conversation space* (set in the town's tavern). The actions taken by NPCs in the world simulation populate the content of an NPCs' dialogue in the following conversation. Players can provide information or suggestions to NPCs in the tavern's conversational space which alter what actions they want to take in the simulated world.

The world simulation can be broken into roughly three subsystems: building the pool of possible actions available to each NPC (the *Action Builder*), selecting which of those possible actions to take (the *Action Heuristic Manager*), and executing and recording the results of the selected action (the *Action Execution Manager*). The Action Builder aggregates all actions provided by features in a location and begins the process of binding them to the acting NPC. The Heuristic Manager weighs how much that NPC wants to perform each of those actions against their goals and picks from those with the highest weight for the NPC to perform. The Action Execution Manager realizes the selected actions in the simulated world state. In cases where the action has variable results, the result is selected by this system.

The conversation space can roughly be broken up into two subsystem: a system for selecting the social moves available to the player under the current context and for choosing the next response for the NPC (the *Conversation Engine*) and the system for turning the system representation of this data into English text understandable by a human player (the *Social Move Verbalizer*). The Conversation Engine in turn can be thought of as three parts, one handling aggregating potential social moves available to the player to choose, one handling choosing a response for the NPC given player input, and one which curates which world facts are included in NPC social moves.

We shall now go into each of these systems in detail.

3.1 World Simulation

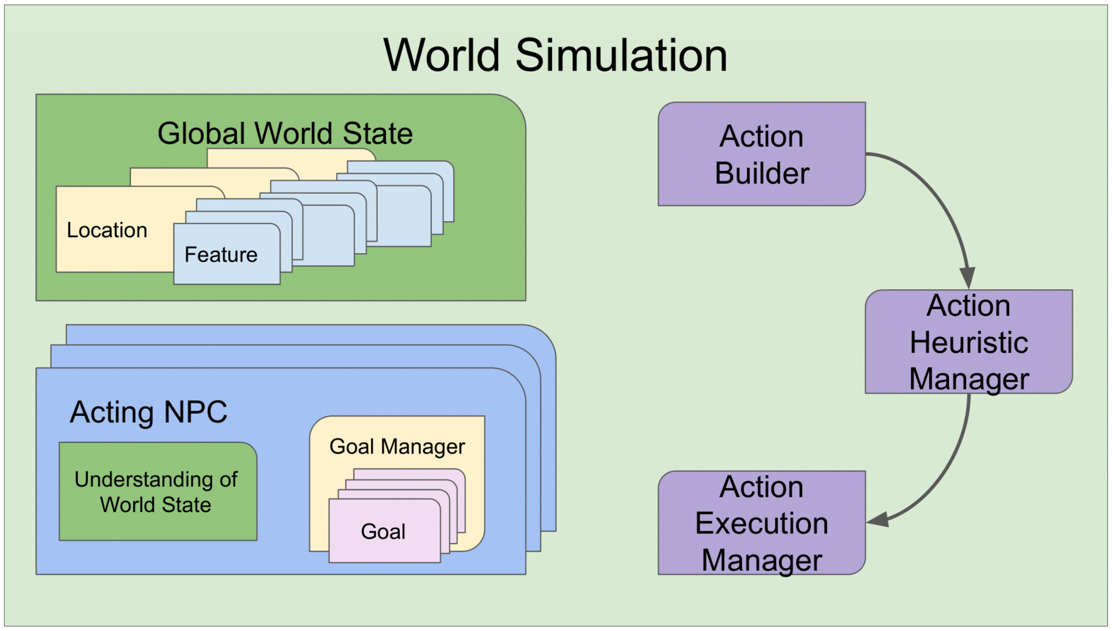


Figure 3.2: An overview of the World Simulation system parts. Important pieces of data include the current global world state (in dark green) and the list of NPCs (in blue). The functions of world simulation is broken up into three subsystems: the action builder, the action heuristic manager, and the action execution manager (all pictured in purple). How these systems interact with each other and the general state of world simulation will be discussed in their specific sections.

Before we get into how actions are chosen and executed, we first describe entities in the simulation and the structure of the world being simulated. That world is made up of discrete “locations”. These are general areas, for example, the town, the forest, or the farm. Any action taken within a given location is seen (registered in the knowledge base) by all NPCs also in that location at that time or who entered or exited

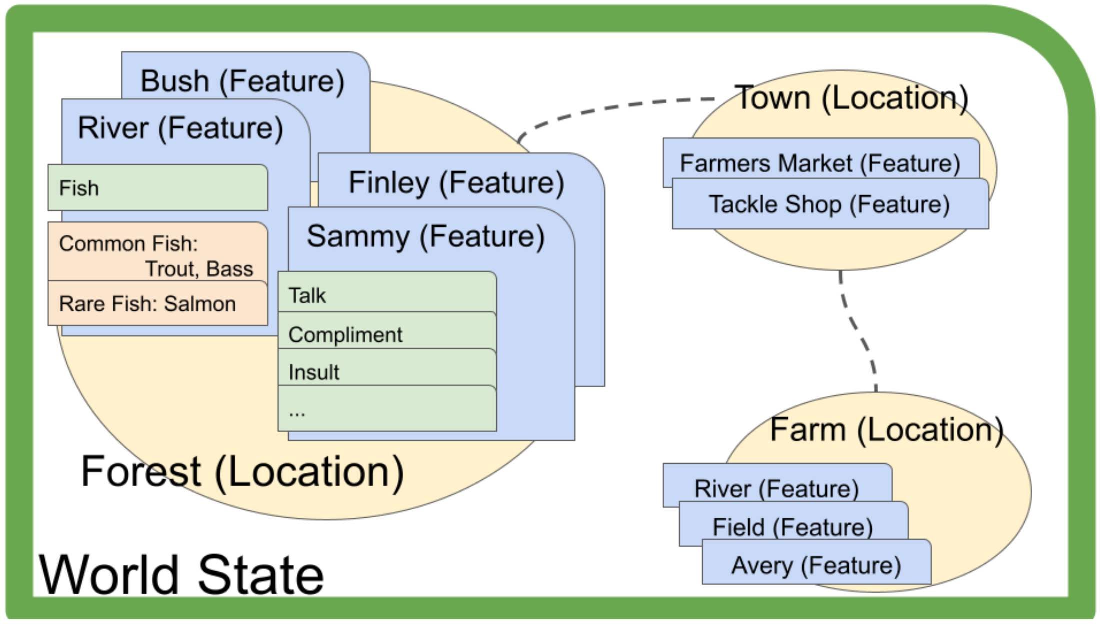


Figure 3.3: Here we look more closely at the contents of World State. It is a collection of Locations (yellow circles), each of which contains a number of Features (in blue). Notice that the NPCs (Finley, Sammy, and Avery) are treated as features for the purposes of world state. Additionally, each feature has a set of actions they provide (shown in green) and a collection of resource lists (shown in light orange). In addition to the features depicted here, each location has a "door" feature for each location it is connected to which provides NPCs with the move action (in this diagram these features are represented as the dotted lines connecting locations).

that location at that time.

Within each of these locations are "features". Each feature has a collection of actions that can be performed on them. For example, in the forest location, there is a river feature that allows NPCs to perform the fishing action there.

In addition to keeping track of what actions they provide, features also hold lists of resources their actions might provide. For example, all river features provide the fishing action for NPCs as well as hold a list of fish that can be caught at them specifically. The river in the forest might provide bass, trout, and salmon, while the river in the meadow might provide trout and carp. NPCs by default initially do not know the contents of these resource lists, but rather discover them through the results of actions. For the purposes of this demo, each NPC has a domain of actions (related to their profession) for which they know already know the relevant resources. For example, the fisherman knows what fish are in the river and the farmer knows what plants are growing on their farm.

Specific actions are not unique to specific types of features either. In the current world description, both “brush” features and “meadow” features provide the forge action, while the resources yielded are dramatically different (one providing primarily flowers, the other primarily berries). One could imagine a larger game having lake, river, and ocean features, for example, and all three providing the fishing action while yielding different catches, as well as accompanied by other actions specific to those bodies of water (perhaps a “diving” action at lakes or a “play in the surf” action at oceans).

Features can roughly be divided into two classes: environment and people. Yes, people. The NPCs in the simulation are actually treated largely the same as things like rivers and shops, at least from an action selection point of view by their peers.

Each NPC is represented as a feature in the world view of the others. Unlike features like rivers, other NPCs can move from location to location. But, like rivers, NPCs provide their own set of actions for other NPCs to take toward them. These are all social actions, like talking or gift giving, but at an abstract level work exactly the same as actions taken on non-sentient features.

As part of that world state, NPCs have opinions about each other. These relationships are tracked over two axes, romantic and platonic, and are not symmetrical. For example, Finley might have a negative, platonic opinion of Sammy, while Sammy has a neutral, platonic and a positive romantic opinion of Finley (Figure 3.4). Additionally, characters track what they think other NPCs think of them. For example, Sammy might think that they and Avery mutually consider each other friends, when in fact Avery additionally likes Sammy romantically (Figure 3.5). In theory, NPCs can keep track of what other NPCs think of each other as well, but this is currently not used.

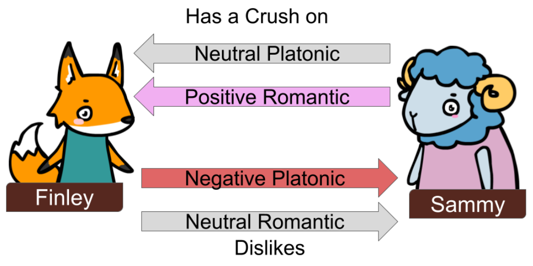


Figure 3.4: Characters can hold non-symmetrical opinions of one another over two axes. Here, Finley dislikes Sammy platonically and is indifferent to them romantically, while Sammy likes Finley romantically.

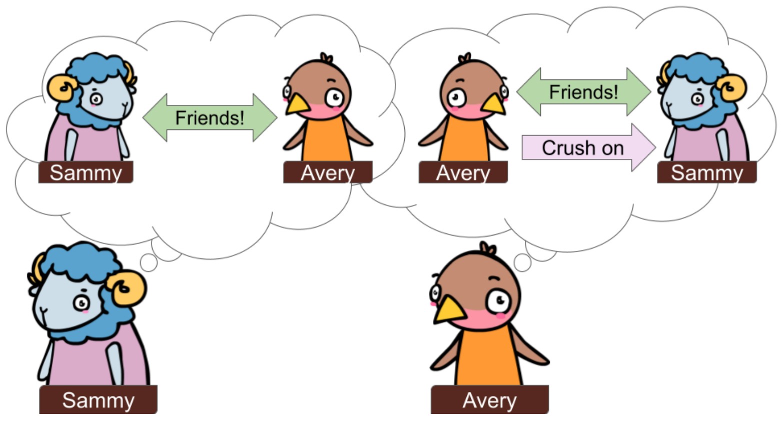


Figure 3.5: Characters can believe other characters have opinions of them and these belief do not always match. Here, Sammy believes that their relationship with Avery is purely platonic, however, we can see Avery actually has a crush on Sammy.

In addition, features have a table of active status effects. Status effects are composed of a type (“happy”, “sad”, or “angry”), a strength (numerical), and a duration (measured in simulated time steps). Additionally, they can optionally be directed toward a particular NPC. Originally, there were going to be status effects that could be applied to environment features as well, such as “broken”, but our demo only implemented emotional status effects to be applied to NPC features. These are added to NPCs via the effects of actions and wear off over time. Each type of emotional status effect temporarily modifies the opinion NPCs have of each other. For example, “happy” increases the platonic opinion the happy NPC feels toward other NPCs. This effect is stronger for the target of that happy status.

Outside of world state, NPCs are represented as autonomous-agents, and each has their own copy of world state. This is not a perfect copy of world state, but rather represents their knowledge of world state. In general, the true state of the simulated world will be referred to as “global world state”, while an NPC’s copy will be referred to as their “understanding of world state”.

In addition to an understanding of world state, NPC agents also keep a record of executed actions they are aware of. These are actions they either participated in, witnessed, or which the player told them about.

With the organization of entities involved in the world simulation out of the way, we can now look at the flow of action selection. Every day, each NPC starts in their home at 8 am. Every half-hour (simulated world time, not physical, real-world time), the NPCs collect their available actions from the Action Builder system and assess the desirability of each action against their list of goals in the Action Heuristic Manager. These goals in question are generated from the NPC's understanding of the world state in their Goal Manager. After weighing each available action against their current goals, they choose (with weighted probability) from their top five weighted actions. This action is then executed in the Action Execution Manager. Then the next NPC repeats this process until all three have acted in that time step. This is repeated every half-hour from 8 am until 10 pm, when the day ends.

We will now go into each of these systems in greater detail.

3.1.1 Goal Management

Before an NPC can begin deciding what action to take this time step, they first need to evaluate what their goals are. NPC goals in this system are not a static set of objectives. Rather they change according to the context of the world. And since the state of the world changes with every action taken, these goals must be regenerated before every action is selected.

At the highest level, all NPCs have the same set of "goal modules" which specify a prerequisite and a collection of goals to pursue if that prerequisite is true. For example, a common goal for the NPCs of this demo is to want to date another NPC. In truth, lying dormant in all the NPCs is three separate goal modules per other character to date that character. In other words, Sammy has six goal modules related to dating–three to date Avery, three to date Finley. Each of those three modules per character has a different level of priority (low, medium, or high, depending on Sammy's opinion of their prospective romantic partner). Additionally, all six of these modules have prerequisites such that at most only one of these modules will be active at a time. For these goal modules, the prerequisites look something like:

- Actor romantically likes Target a certain amount (the exact value is different for the three versions, with a higher value here paired with a higher goal priority)
- Of all the other NPCs, the Actor romantically likes the Target NPC the most
- The Actor and the Target are not currently dating.

This allows the NPC's goals to change to reflect their current opinions or understanding of world state. For example, although the game begins with Sammy in love with Finley and so wanting to date them, it is possible for the player and/or Finley to damage Sammy's opinion of Finley badly enough that Sammy doesn't romantically like Finley anymore. At that point, Sammy's goal to date Finley would deactivate and so dating Finley would no longer be a top-level goal for Sammy. Simultaneously, perhaps Avery will improve Sammy's opinion of them. In that case, Sammy's goal to date Avery may activate, making that among their top-level goals.

In this demo, other goal modules include:

- Becoming best friends (active when already have friendship relation at a certain value or higher)
- Becoming mortal enemies (active when already have friendship relation at a certain value or lower)

- Improving skills (active if they have a particular profession)
- Eating food they like (when they have that food in their possession).

These are all considered top-level goals and direct their general behavior.

From these top-level goals, additional subgoals are created from the preconditions of actions or probability modifiers of outcomes that would help progress that goal. Preconditions require certain world state values, while probability modifiers (which we will be discussing in more detail in the Action Heuristic Manager section) result in higher probabilities if those values are above certain thresholds. These values include:

- The number of inventory items
- The relationship value with another NPC
- The skill level of the NPC.
- The number of inventory items
- The relationship value with another NPC
- The skill level of the NPC.

As such, from preconditions and probability modifiers, we can generate subgoals to get these values within a given range.

For example, let us continue the previous dating goal example. Let's call the actor here Avery and the subject of their affections Sammy. The only action which makes the "dating" state occur is the "ask out" action. In particular, it is only the "successful" outcome of that action that results in two characters dating. So, the probability modifier for that outcome (that starting to date is more likely if Sammy romantically likes Avery more) becomes the new subgoal "increase Sammy's romantic opinion of Avery".

This goal can be achieved in a number of ways, but if Avery knows that Sammy loves strawberry cake, one action they might want to take to progress the goal of increasing Sammy's opinion of them is the "give_strawberry_cake" action. This action has the precondition that one has a strawberry cake to give, which results in the sub-goal "have at least 1 strawberry cake in Avery's inventory".

This too can be unrolled into further subgoals, to have a cake, one might bake one. To bake a cake, one needs a collection of ingredients such as flour and strawberries. To get strawberries, one might forage in the forest. In this way, a great many subgoals are generated from a small collection of top-level goals, allowing for great variability in the desired action on the part of NPCs.

However, not every precondition or probability can be altered by the acting NPC. Things such as modifying the contents of another NPC's inventory or an item being of a particular type (food or not food for example) are considered "non-actionable" and do not generate goals for NPCs. These are filtered out by the system via a set of authored constraints. Actions that have such preconditions that are not currently evaluated true are not included at all in the goal unrolling process.

Some goals, however, have no actions the NPC is aware of that will advance them. These are called "stuck goals", and their detection allows the player to ask the question "What are you having trouble with?" during conversation. These occur when an NPC is either unaware of where an item could be found, unaware of what another NPC's item preferences are, or when all the actions that would advance the goal have a non-actionable precondition.

3.1.2 Action Builder

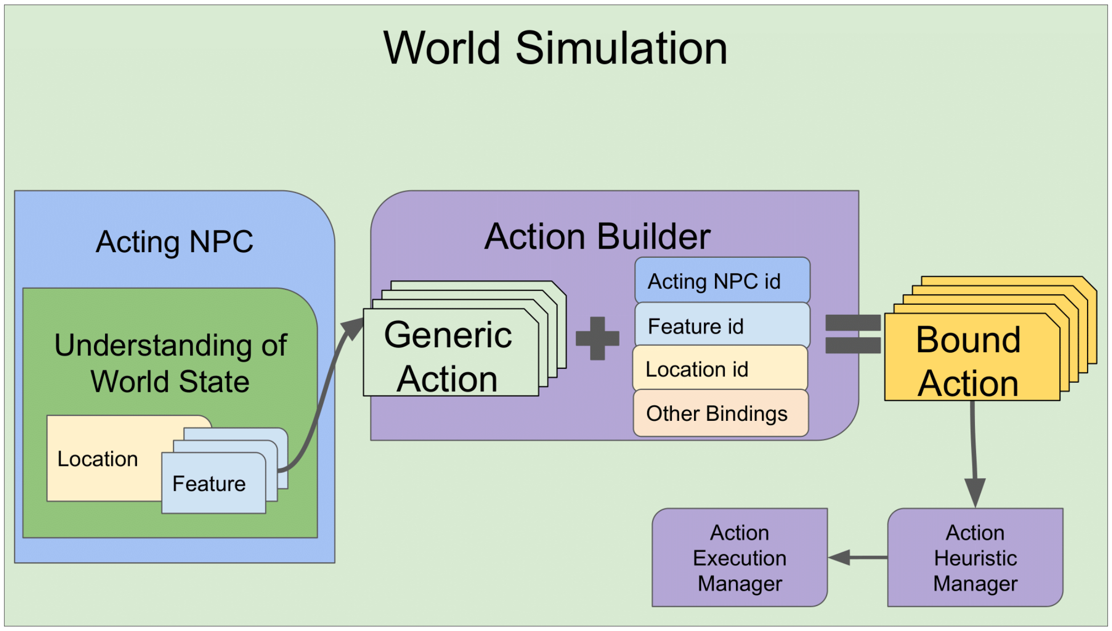


Figure 3.6: The Action Builder is the first step in action selection in the World Simulation system. It takes the generic actions the acting NPC knows about from the features in the location it is currently in and creates a set of bound actions for the Action Heuristic Manager to evaluate by binding the actor, feature being acted upon, location, and action specific bindings (such as items if one is involved in the action).

Actions go through a number of steps before becoming a remembered event in the minds of the NPCs. The first of these is to go from generic descriptions of abstract actions to specific "bound actions". Generic actions are made up of:

- the list of potential outcomes

- any hard preconditions that must be honored
- a list of fields that will require further specification (unbound bindings)

When all is done, a fully realized and remembered action is made up of:

- who took the action (the actor)
- what they acted on (the feature)
- where the action took place (the location)
- when the action took place (in world time)
- the reasons the NPC took the action in question (weighed against their goals)
- a list of rejected and invalid alternatives
- a list of realized effects (the actual results of the action)

It is up to the Action Builder to take these generic actions and make their first transformation, turning them into bound actions. In this step, the Action Builder adds the actor, the feature, and the location to the generic action, as well as specifying some of the bindings specific to the action.

For example, take the action "fishing". The generic version looks something like Figure 3.7. Here we can see that there is very little specified about it. We can see there is one precondition, that something called "#a#" has an item "fishing_pole" in their inventory. This action is not a valid option for an actor to take if this condition is not true. But what is "#a#"?

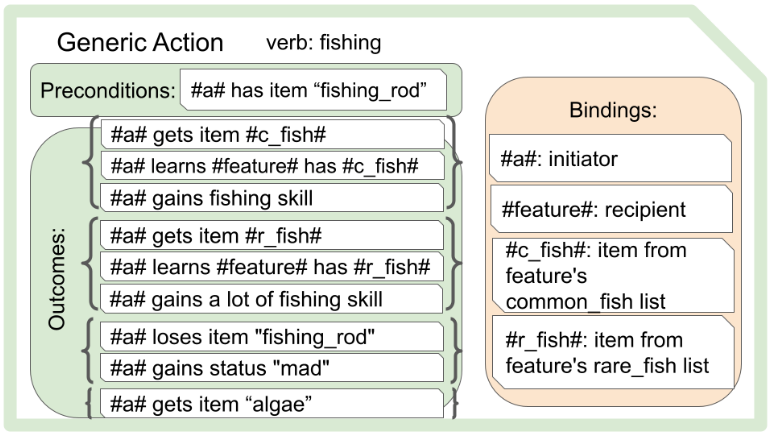

Generic Action
verb: fishing
Preconditions:
#a# has item "fishing_rod"
Outcomes:
#a# gets item #c_fish#
#a# learns #feature# has #c_fish#
#a# gains fishing skill
#a# gets item #r_fish#
#a# learns #feature# has #r_fish#
#a# gains a lot of fishing skill
#a# loses item "fishing_rod"
#a# gains status "mad"
#a# gets item "algae"
Bindings:
#a#: initiator
#feature#: recipient
#c_fish#: item from feature's common_fish list
#r_fish#: item from feature's rare_fish list

Figure 3.7: Here we have an example of a generic action: fishing. It has a precondition, that the actor must have a fishing rod in their inventory. It has four potential outcomes: one in which the actor catches a common fish, one in which the actor catches a rare fish, one in which the actor breaks their fishing rod, and one in which the actor catches trash ("algae"). Who the actor (initiator) is, what feature is in use, and what fish they catch needs to be bound before this becomes an executed action.

To answer that, let us look at the list of bindings. Here we see another reference to "#a#" specifically "#a#: initiator". This means that whoever initiated this action (the actor) is the entity that should fill the role of "#a#" in this action. There are six roles that can be specified here with special meaning:

- "Initiator" - the actor
- "Recipient" - the feature being acted upon
- "Bystander" - any person in the same location who is neither initiator or recipient
- "Any" - any person anywhere in the system regardless of location
- "Location" - the current location
- "Location_any" - any location in the world

However, in practice, only the first two are used in this demo.

This is one of three kinds of bindings, an entity binding. We can see a second kind (a string binding) in “#c_fish#: item from feature’s common_fish list”. Here we associate the string “#c_fish#” with the resource category “common_fish” (always from the resource list provided by the feature being acted on in the action). The third kind of binding is an inventory item binding, which we shall come back to in a moment.

Back to our example, this means that our precondition says “the actor must have a fishing pole” and the result of this action will be one of four outcomes: our actor might get an item from this feature’s “common_fish” list, get an item from this feature’s “rare_fish” list, lose the required item “fishing_rod”, or catch “algae” (generally considered trash in this kind of video game).

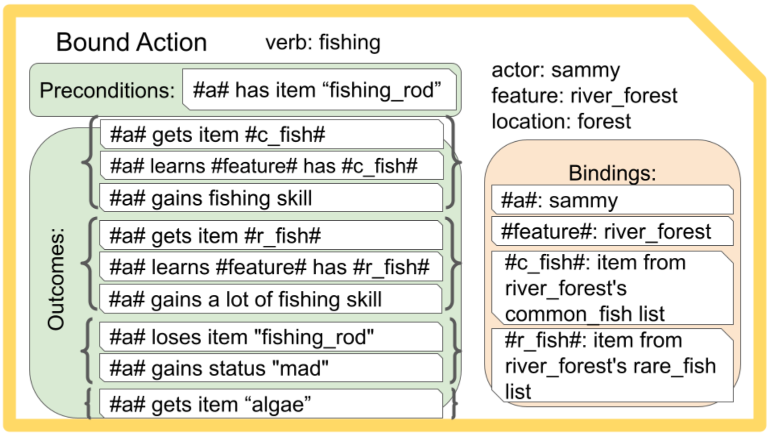

Bound Action
verb: fishing
Preconditions:
#a# has item “fishing_rod”
actor: sammy
feature: river_forest
location: forest
Outcomes:
#a# gets item #c_fish#
#a# learns #feature# has #c_fish#
#a# gains fishing skill
#a# gets item #r_fish#
#a# learns #feature# has #r_fish#
#a# gains a lot of fishing skill
#a# loses item "fishing_rod"
#a# gains status "mad"
#a# gets item “algae”
Bindings:
#a#: sammy
#feature#: river_forest
#c_fish#: item from river_forest's common_fish list
#r_fish#: item from river_forest's rare_fish list

Figure 3.8: Here we have a bound action created from the Action Builder. Notice that, despite being called a bound action, not all the bindings have actually been bound yet. Rather, only things that the NPC has control over (who is acting and on what) are bound. The NPC, as much as they might want certain outcomes, has no control over what fish they catch. This kind of binding is left unbound until the the Action Execution Manager determines the results of the actions.

Now we can compare this generic action to the bound action which the Action Builder might have created for Sammy while they stood in the forest location (Figure 3.8). We can see there are new fields now, the actor, feature, and location. Additionally, some of the bindings have changed. Some have gone from descriptions of what each binding could be to specific entities. Sammy (the actor) is now bound to the role of "#a#". The feature has been filled into the "#feature#" binding.

But why is "#c_fish#"'s binding not specified at this time?

At a glance, it might seem that we should be able to specify the items here. We know what feature is being acted upon, after all. We have access to that feature's list of provided resources. We can, right now, look up what fish are considered "common_fish" here.

We could, except for the fact that the NPC for whom we are building this action may not know what those items are. Or they might only know about some of them. This is due to the fact that NPCs each maintain their own list of resources available at each feature in their understanding of the world state. In the next step, the acting NPC will be evaluating whether or not this is an action they wish to take based on their own understanding of the world. However, in the action execution manager, we will want to select the fish caught based on the true state of the world (the global world state) instead. This means we cannot make this binding now, as picking from the NPC's understanding of the world would not reflect the nature of the world for the action execution manager. Further, we cannot pick from the true world state's resource list as that would give the NPC too much information when they evaluate whether they want to perform this action or not in the action heuristic manager.

In this example, the generic action to the bound action is a one-to-one transformation. Of the bindings used, there is only one entity that can be slotted into each binding. However, this is not generally true. In cases where the “bystander” or “any” bindings are used, there are a number of potential entities that could be slotted into that binding. Each combination of bindings would result in another bound action.

This is particularly noticeable in the “give_#item#” action. This action has the third kind of binding, the inventory item binding. Here every item in a specified entity’s inventory is a potential binding. In the example of “give_#item#”, the actor’s inventory is the one considered. As such, a bound action is created for each inventory item that could be given.

This is the Action Builder’s job. It collects all the features in the same location as a particular NPC and generates all bound actions available to the acting NPC at that time. At a glance it might seem that this could be done once in the beginning, however, because it is possible for features to move (remember other NPCs are considered features as well) or for entities’ inventory contents to change, this must be recreated for each NPC every time step.

3.1.3 Action Heuristic Manager

Once the Action Builder has returned all the possible actions available to a given NPC, the NPC then needs to decide which of those actions to perform. This task is handled by the Action Heuristic Manager. This manager takes each bound action

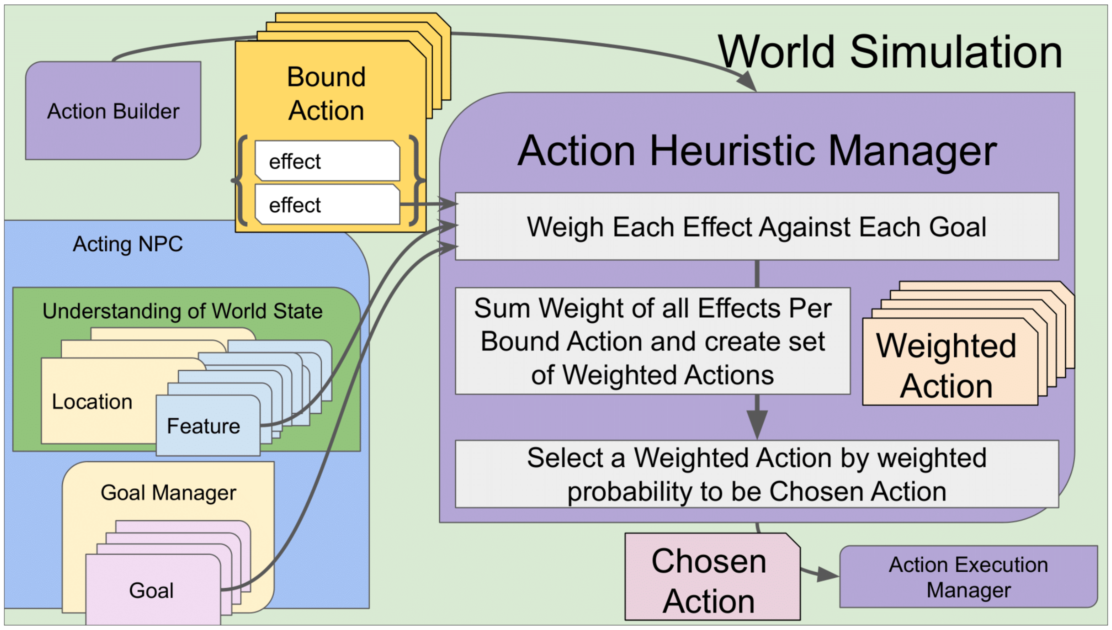

World Simulation
Action Builder
Bound Action
effect
effect
Acting NPC
Understanding of World State
Location
Feature
Goal Manager
Goal
Action Heuristic Manager
Weigh Each Effect Against Each Goal
Sum Weight of all Effects Per Bound Action and create set of Weighted Actions
Weighted Action
Select a Weighted Action by weighted probability to be Chosen Action
Chosen Action
Action Execution Manager

Figure 3.9: From the Bound Actions built in the Action Builder, the Action Heuristic Manager creates a set of Weighted Actions by first weighing how much an NPC wants each effect of each potential outcome to happen based on their understanding of the world state and their list of goals, then summing those values to get a weight for each action. From those weighted actions, one is selected from the top five by weighted probability. This Chosen Action is passed to the Action Execution Manager.

and weighs their potential results against each of that NPC's current goals. It then outputs a weighted action for each bound action, containing a numerical representation of their desire to perform that action as well as a list of the reasons for this weighting. In addition to returning a list of weighted actions, it also returns a list of all actions the NPC is currently unable to perform despite the features in the room providing a place to enact them, for debugging purposes.

In more detail, the first step in this process is to filter out the actions that the NPC is unable to perform based on the action's preconditions. For an example, consider the fishing action discussed earlier (see Figure 3.8). Here the precondition is that the actor must possess an item called “fishing_rod”. For any NPC who does not meet this criterion, this bound action gets added to the list of invalid actions. One can create preconditions on a variety of attributes, from inventory content, to inter-NPC relationship values, to recent activity.

Once determined to be a valid action, each of the action’s potential outcomes is evaluated against the NPC’s goals. This is a good time to look at outcomes in greater detail. An outcome is made up of a chance of it occurring and a list of effects (Figure 3.10). The chance of occurring can be a static probability or a dynamic value calculated based on world state. These dynamic probabilities consider world state values such as NPC relationships, inventory item counts, or skill level and compare these values to a specified range, evaluating to 100% if the value is equal to or higher than the upper bound, 0% if the value is equal to or lower than the lower bound, and scaling linearly along that range. Additionally, these values can be multiplied, added, or inverted to create a wide range of dynamic probabilities for outcomes.

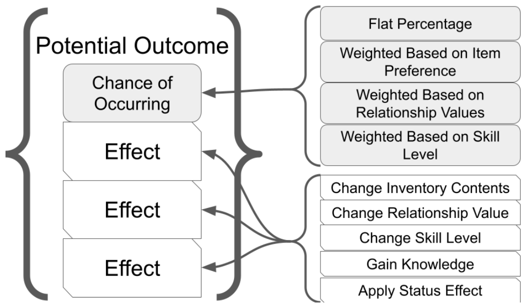
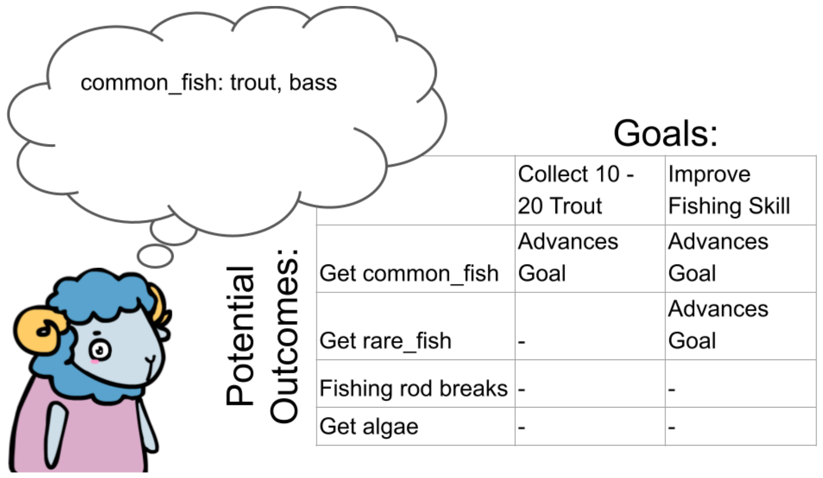

Potential Outcome
Chance of Occurring
Effect
Effect
Effect
Flat Percentage
Weighted Based on Item Preference
Weighted Based on Relationship Values
Weighted Based on Skill Level
Change Inventory Contents
Change Relationship Value
Change Skill Level
Gain Knowledge
Apply Status Effect

Figure 3.10: Outcomes are made up of a *Chance of Occurring* and a list of *Effects*.

In the previously mentioned fishing example, the fishing action has three outcomes which could generally be summarized as: catching a common fish, catching a rare fish, or the fishing pole breaking. In the first of these outcomes, the individual effects are: catching a fish, learning that fish can be found in that river, and increasing skill.

These effects are what determines whether an action serves a particular goal or not. The desire for any given effect is calculated based on how well the effect advances the NPC's progress toward a goal multiplied by the weight of the given goal, summed over all the NPC's goals (though in practice it is unusual for an individual effect to advance more than one goal at a time). The sum of an NPC's desire for each effect to occur is then multiplied by the chance that outcome will occur, then added to the values obtained by the other potential outcomes.

Additionally, each pair of goal and effect that progresses that goal are recorded as part of the "reason" this action is desirable. The NPC does not do anything with this reasoning in the world simulation, but it is useful debugging information as well as used to answer certain questions during the conversation with the player.



	Collect 10 - 20 Trout	Improve Fishing Skill
Get common_fish	Advances Goal	Advances Goal
Get rare_fish	-	Advances Goal
Fishing rod breaks	-	-
Get algae	-	-

Figure 3.11: In this example, Sammy knows that the river they are thinking about fishing at has trout and bass available as common_fish.

To illustrate how this works, imagine that Sammy knows that the forest river has trout and bass as "common_fish" yields and their goals include "collecting between

10 and 20 trout" and "improving the fishing skill". Let us also assume that Sammy has a fishing rod (Figure 3.11). At this stage, the bindings would be bound according to the NPC's knowledge of the world. In this example that would be binding "#c_fish#" to the list of known yields, "trout" and "bass". This means Sammy would want the first outcome because there is a chance to catch trout bringing them closer to the goal of having a certain amount of trout, and it will increase their fishing skill lining up with their goal of improving that skill. Similarly, they would want the second outcome because it would increase their fishing skill. Neither the third outcome (breaking the fishing rod) nor the fourth (catching algae) would contribute to their desire to perform this action. It is also possible for performing an action to be an NPC's goal. In this case, the careful evaluation of the outcomes and their effects are skipped in favor of comparing the action ID of the potential action and the goal action.

The final step taken by the Action Heuristic Manager is to choose one of these weighted actions for the NPC to perform. The top five weighted actions are considered and chosen via weighted probability. This "chosen action" contains all the information of its weighted form plus the list of all invalid actions discovered in the initial stages of this process and a list of all unselected ("rejected") actions that were weighted but not selected in this last step. These lists are both used exclusively in debugging currently, however, their use in the conversation side could be imagined (if the NPCs brought up being conflicted about what they wanted to do but then did not, for example).

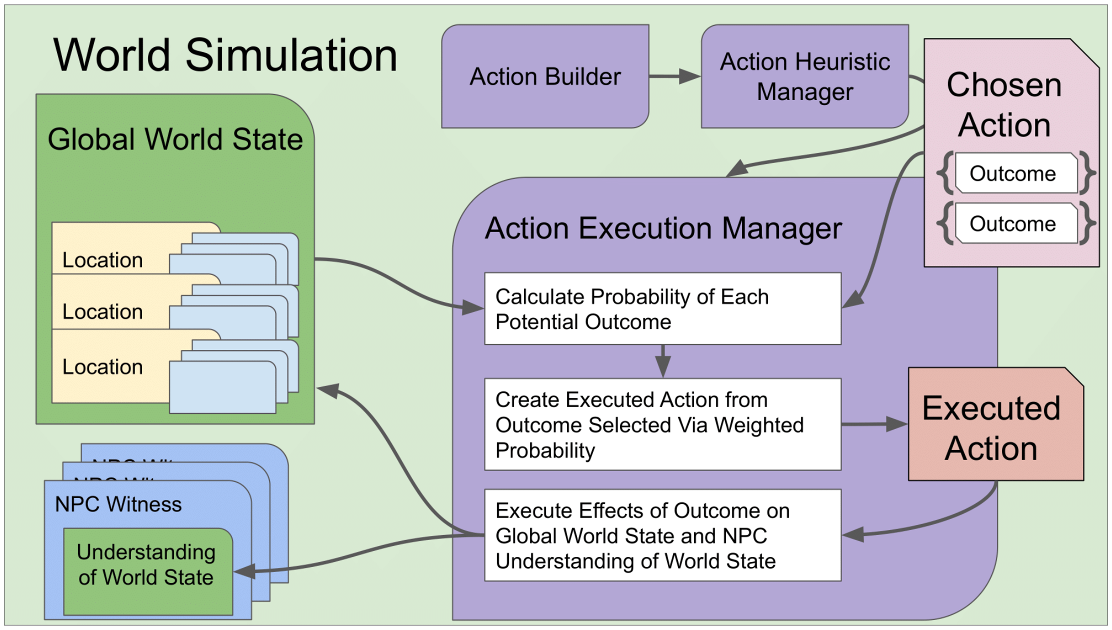

World Simulation
Global World State
Location
Location
Location
NPC Witness
Understanding of World State
Action Builder
Action Heuristic Manager
Chosen Action
Outcome
Outcome
Action Execution Manager
Calculate Probability of Each Potential Outcome
Create Executed Action from Outcome Selected Via Weighted Probability
Execute Effects of Outcome on Global World State and NPC Understanding of World State
Executed Action

Figure 3.12: The Action Execution Manager takes the chosen action created by the Action Heuristic Manager and creates an Executed Action from one of the chosen action's outcomes (chosen by weighted probability based on each outcome's probability modifier). The effects of that outcome are then applied to Global World State as well as the Understanding of the World State of any NPC who witnessed the action.

3.1.4 Action Execution Manager

This chosen action is then sent to the Action Execution Manager. Here, the probability of each potential outcome is calculated (the probability modifiers mentioned earlier). Once these values are calculated, one outcome is selected via weighted probability.

Next, based on the outcome, a single entity or item is specified for any unbound variables. In the case of the fishing example earlier, "#c_fish#" might be bound to specifically "trout" at this stage. Everywhere in the effects, the reference to "#c_fish#" would now evaluate as "trout".

The effects of the selected outcome are then run, updating the true world state as well as the knowledge of that world state for any NPC who was around to witness the action (see the introduction to this section [Section 3.1] for detail on what kind of state this might include). In addition, a record of this event, now an executed action (Figure 3.13), complete with the time it was executed and a list of the realized effects, is added to the list of remembered events for all NPCs who witnessed the event.

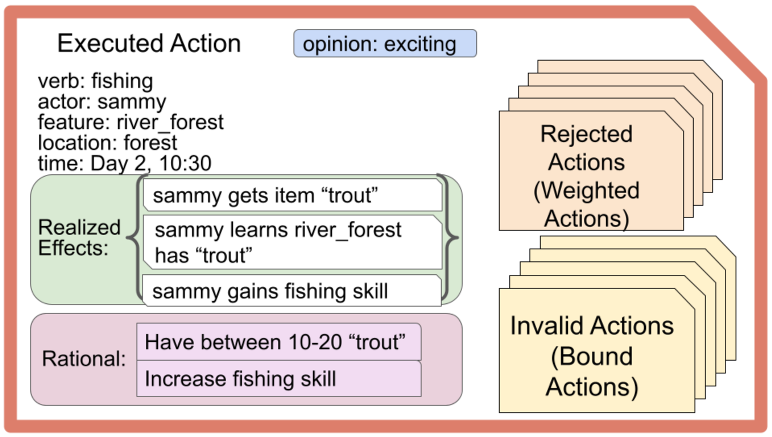


Figure 3.13: An executed action.

After actions are fully executed, the actor forms an opinion about how they feel about that action based largely on what they thought the results of the action were going to be versus what the actual results turned out to be. This is done by running the fully executed action through the Action Heuristic Manager again and comparing the weight of the executed results to the weight calculated earlier when they were deciding which action to take. If they did not get the outcome they were hoping for they are disappointed, if they got it (or something even better they did not know about) they consider it exciting. This opinion is used during conversation to filter what actions

NPCs tell the player about when reporting their day.

3.2 Conversation Space

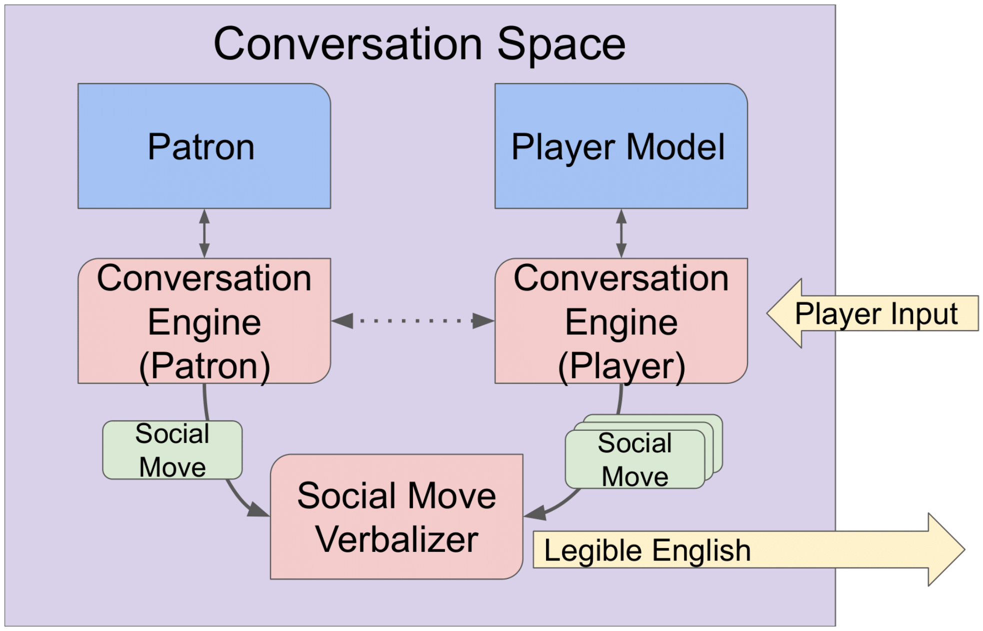


Figure 3.14: Conversation Space is made up of two Conversation Engines (one for the player and one for the NPC patron) and the Social Move Verbalizer. The Conversation Engines select a set of appropriate social moves given the current context. The Social Move Verbalizer turns the abstract social moves into readable English output to be displayed to the player. The Conversation Engines do not talk to each other directly, but the social move that is outputted by one is passed along to be the input for the other.

Crosston Tavern's play takes place in the form of conversations with NPCs. The player and the NPCs take turns taking actions, called "social moves" to differentiate them from the actions taken in the world simulation as these are structurally different. Each social move consists of an abstract description of the move being made and, optionally, a list of references to world state (referred to as "facts") being conveyed to their conversation partner. For example, when the NPC tells the player about their day they are taking the "tellAboutDayEvents" social move accompanied by a collection of discrete events from the most recent day. We shall discuss the structure of these social moves and the facts they may contain below.

Which move (or moves) is available, with which facts attached, is determined by the Conversation Engine. It has two primary responsibilities: picking the single move taken by the NPCs each turn and providing the list of social move options to the player. Then, once either chosen for the NPC to say or included as an option in the player's menu of dialogue choices, each move is then converted into readable English that the players can understand via the Conversation Verbalizer. We shall discuss both of these systems in more detail in the following sections.

Conversations follow a dependable structure. Play begins with the NPC and the player greeting one another. Here the NPC can characterize their day as good, frustrating, sad, or uneventful. Food is then ordered, either spontaneously or on (or against) the recommendation of the barkeep. The NPC then comments on the food, mentioning if it is a favorite dish or if they like the ingredients in it or not. At that point, conversation opens up. A wide selection of social moves are made available to the player, from asking what the NPC did, to what their goals are, to what they think of other characters. Facts included in social moves are learned by the listener, updating the listener's understanding of their simulated world. This process is symmetrical between the player and NPC, as both store their understanding of world state in an identical manner. Through this modeling of player knowledge, we can include among their dialogue options statements about facts the player has learned about. Through these statements, the player can subtly influence the actions of other NPCs. For more direct control, the player also has the ability to suggest new goals for the NPCs to pursue, allowing the player to push the world state in wildly varying directions.

3.2.1 Social Moves

As mentioned previously, social moves are the discrete actions that make up play in Crosston Tavern. Each social move can be thought of as an abstract representation of a line of dialogue. Examples include moves such as “greet”, “tellAboutDayEvent”, or “askAboutGoals”.

Every social move has the option to additionally include a list of facts about the world the listener should learn (generally referred to as “mentioned facts”), a list of facts that the speaker wants to retract (“retracted facts”), and a list of facts that need to be handled in a more complicated manner on a case by case basis (“complex facts”). These lists of facts fill out the specifics of the social move, allowing the player and NPCs to convey specifically certain events, preferences, or goals.

But what is a fact? In this context, facts represent something that is or was true about the simulated world state. This includes:

- Executed actions
- Goals of NPCs
- Relationships between NPCs

- The locations of resources in the simulated world
- The item preferences of NPCs
- Potential actions that could be taken

Most moves involving facts make use of the mentioned facts field, as in most cases the player or the NPC is trying to share a new fact about their world. This is the case in the previously mentioned example of the NPC telling a player about what actions they took earlier that day. Other examples include the "thank" social move taken after being served food, in which the NPC tells the player how much they like the food just served (and so share their item preference), or the "tellAboutGoals" move in which the NPC lists their current goals for the player (and so shares a list of NPC goals).

However, things like goals and relationships change over time; as such, it is sometimes necessary for the NPC to tell the player that something is no longer the case. This is where the retracted facts list comes into play. For example, to answer the player question "Do you still want to [NPC goal]?" (the social move "confirmGoal#"), the NPC might respond with the "confirmGoal#OutOfDate" social move, with that goal listed on the retracted facts list.

In some cases, one wants the NPC to be able to comment not just on the facts included in the social move, but also what they knew before they were told those facts. This is the case with the "tell#Relation#" move available to the player. With this move, the player can tell their conversation partner what another NPC thinks of them. How the player's partner responds should depend heavily on what they thought the other thought of them in the first place as well as their own opinion of them. However, to include this new relationship view in the mentioned facts would overwrite their previous belief of what the other's opinion of them was.

For example, let's say Sammy initially thinks that Finley hates them. If the player was to tell Sammy, "I think Finley thinks of you as a friend" (the social move "tell(Sammy)Relation(Finley)") we would get a response something like, "They really think we are that close? I'm glad!". This response is derived from checking Sammy's belief of Finley's opinion of them from before the player said anything and comparing it to the opinion that the player just shared about what Finley thinks of Sammy. However, as facts on the mentioned fact list are learned as soon as an NPC hears them, before verbalization, we need a mechanism by which learning the facts in the fact list is delayed.

This is where complex facts come into play. Unlike mentioned facts, these are not automatically learned by the listener. Rather, they can be considered special cases, and each social move that makes use of them has additional logic around their treatment. In this example, the conversation partner's previous beliefs about their relationship with the other NPC is compared to the new relationship view and only after the verbalization using that comparison is done does the NPC properly learn the new relationship fact.

3.2.2 Conversation Engine

Both the player and the NPC in a given conversation have a dedicated conversation engine that is responsible for selecting what social moves are appropriate in the given context. The primary difference between the two versions of this system is how many social moves they return when given input. The Barkeep Engine (the one which handles the player's side) generally returns many social moves for the player to choose from while the Patron Engine (the one that handles the NPC's side) always returns a singular social move for the NPC to say to the player. Additionally, which social moves are generally available to each side is different.

Each conversation engine has a speaker (the player in the Barkeep Engine and the NPC in the Patron Engine) and a partner (the opposite of the above). The main goal of the conversation engine is to generate the set of responses appropriate to their partner's most recent social move (frequently referred to as the prompt). Every social move has its own set of appropriate responses, usually generated from a generic social move and either the speaker's knowledge of world state or the facts attached to the prompt. Rather than describing every possible social move that could be used as a prompt and all associated responses, we will instead discuss some common patterns, first for the Patron Engine, then for the Barkeep Engine.

3.2.2.1 Patron Engine

The Patron Engine always returns a single social move in response to any prompt (Figure 3.15). For every question the player can ask and every statement the player can make, the Patron Engine can generate a response. In most cases, for a given class of social move, there is an appropriate class of social move to use in response. These pairings were authored during the design of this system. For example, for any

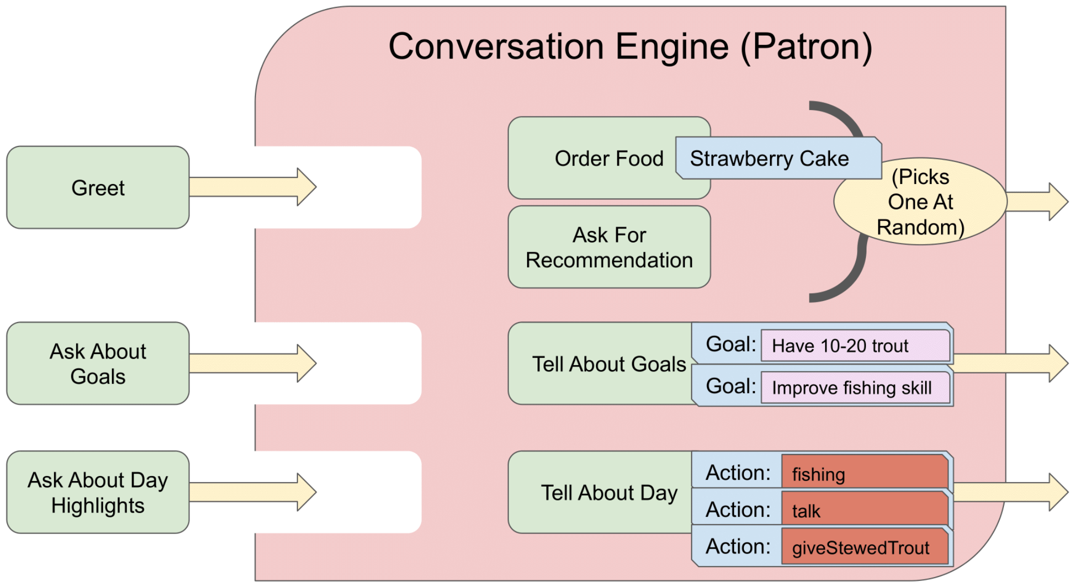


Figure 3.15: The Conversation Engine in determining the next social move for the NPC patron generally takes the player's selected social move as input and selects a specific social move in response. A great many are one to one transformations, as in the case of "askAboutGoals" or "askAboutDayHighlights". A handful have a few potential responses, such as the response to "greet" at which point the NPC can order food or ask the barkeep for a recommendation.

"askAboutGoals" social move, a patron should respond with a "tellAboutGoals" social move.

The real power here is in the facts that the system attaches to these generic social moves. Conversation engines are able to query the state of their speakers to answer player questions. In this "askAboutGoals" example, the system knows not only that the correct response is to "tellAboutGoals" but also that the facts that it should attach should be the goals pulled from the speaker's current goals (Figure 3.16).

In other cases, there is more than one social move that is a potential response. In some cases, as in response to "greet", a social move is selected at random (Figure

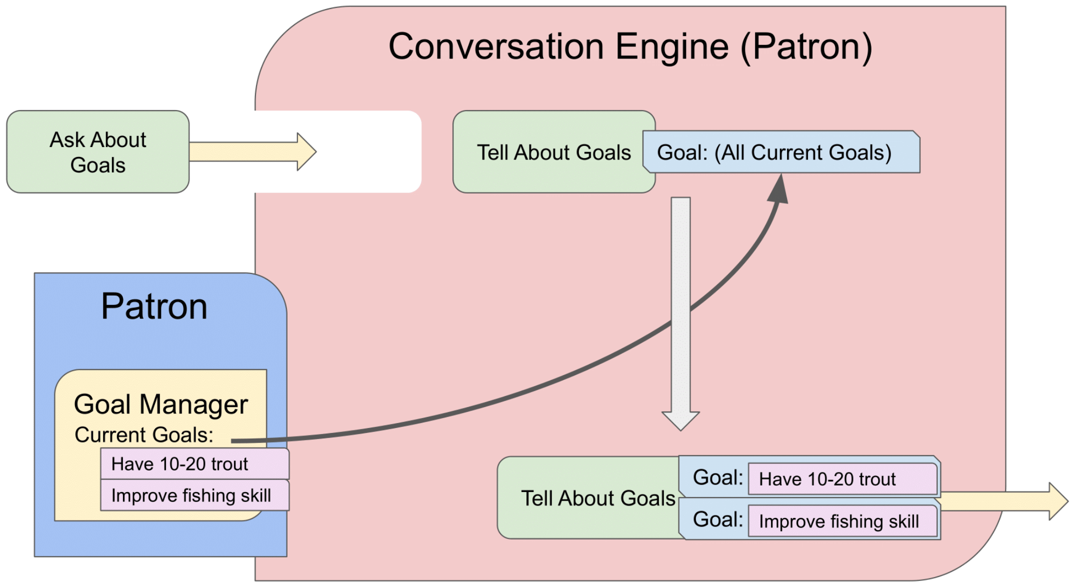


Figure 3.16: Some inputs have one social move output, but which has facts based on the state of the patron. "askAboutGoals" is one such example. Here we can see that it always results in a "tellAboutGoals" response, however, the conversation engine must query the patron's goal manager for the list of goals in question.

3.17). This occurs most frequently when the two potential responses were designed to be equally relevant to the prompt and to provide variety to the NPC's responses.

In other cases, there are multiple responses because there could be multiple reactions to a player's statement or answers to a player's question. For example, when the player tells an NPC about the item preferences of another ("tellPreference" social move) the NPC can either respond that they already knew that fact ("didKnow") or that this is new information for them ("didNotKnow") (Figure 3.17). To determine which response the NPC should give, the Patron Engine queries the NPC's understanding of world state to see whether or not the preference is one they knew about already.

For the most part, when a conversation engine needs to query the speaker's

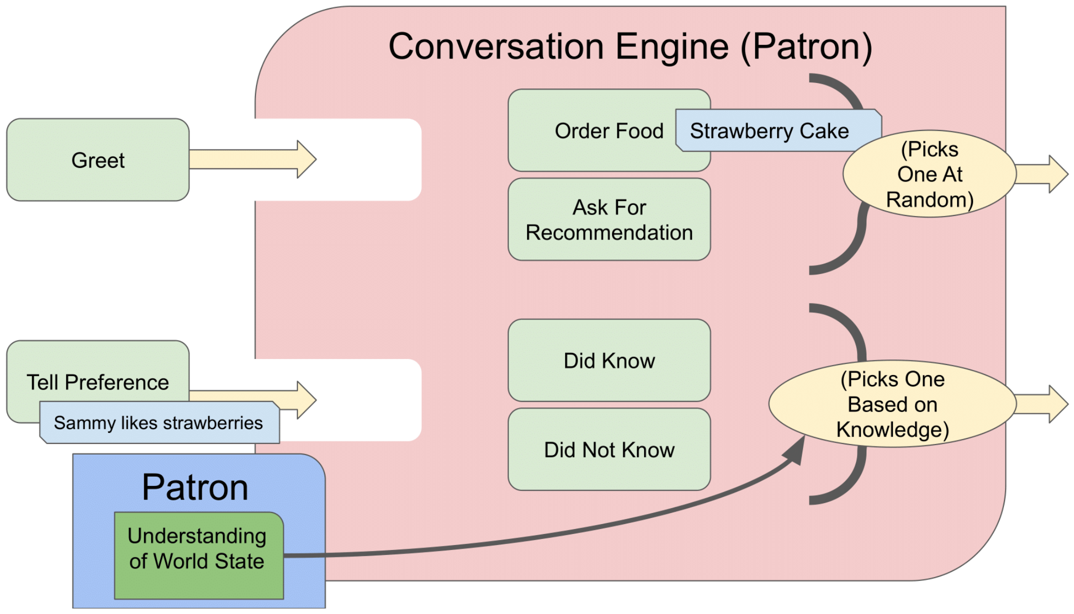


Figure 3.17: In some cases where the NPC has more than one potential response to player social moves, a response is chosen at random, as in the case of "greet". In other cases, the NPC's understanding of the world state is used to respond. This is the case of "tellPreference", where the NPC can respond with either that they "didKnow" the attached fact (that Sammy likes strawberries) or that they "didNotKnow" that.

knowledge, the facts they are looking for are an entire, preexisting list. If an NPC wants to talk about their goals, they list all of their current goals. If an NPC wants to say why they performed an action, they list all of the reasons. If an NPC wants to say why they have a particular goal, they list all its parent goals.

However, if we were to do the same thing for actions taken in a day, it would be completely overwhelming, and frankly, not that interesting. Remember, an NPC's day begins at 8 am and continues until 10 pm. Every half hour they can take an action. That is 28 actions per day that they perform. Additionally, they can observe and report the actions of anyone in the same location as them. Assuming that all three NPCs were in the same location the entire simulated day, that is 84 actions. This is simply not an acceptable answer to “Anything interesting happen today?” So what do we do instead?

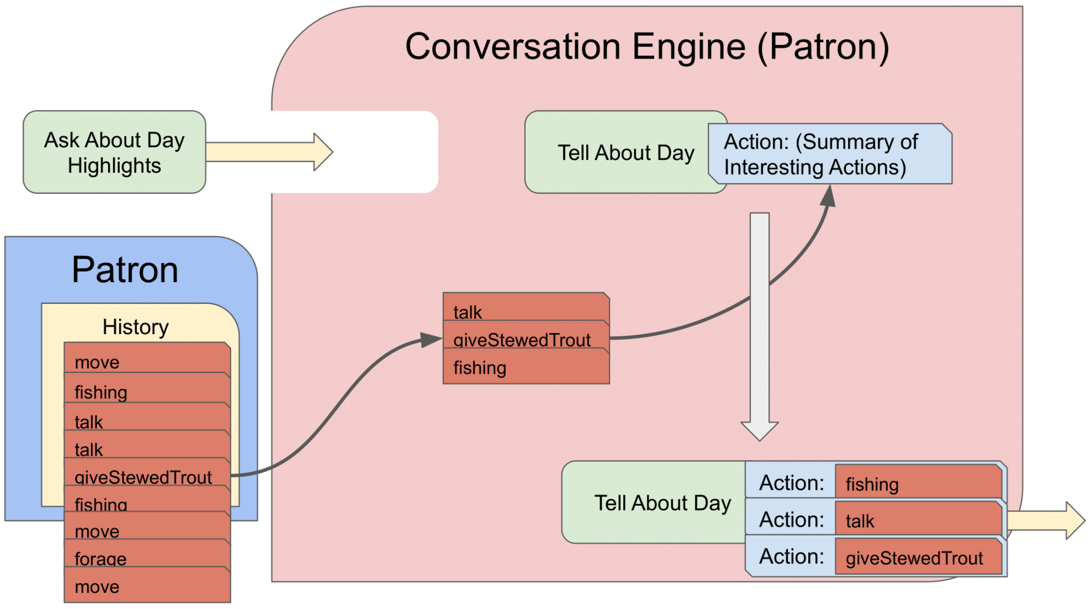


Figure 3.18: The “tellAboutDay” social move’s mentioned facts are pulled from the NPC’s history of past known actions.

Well, first, we remove all movement actions. A fair amount of an NPC’s life is moving from one location to another, and listening to them tell us about how they went from the farm to the forest is just not interesting. This is not to say there is no world in which movement actions could not be parsed into more interesting “super” actions. For example, as a human observer looking through the simulation logs, we were able to notice a fair number of movement actions where one NPC followed another, which was interesting. However we chose not to pursue detecting these kinds of super patterns at this time.

Next, we condensed all repeated actions into one action. So for example, if a

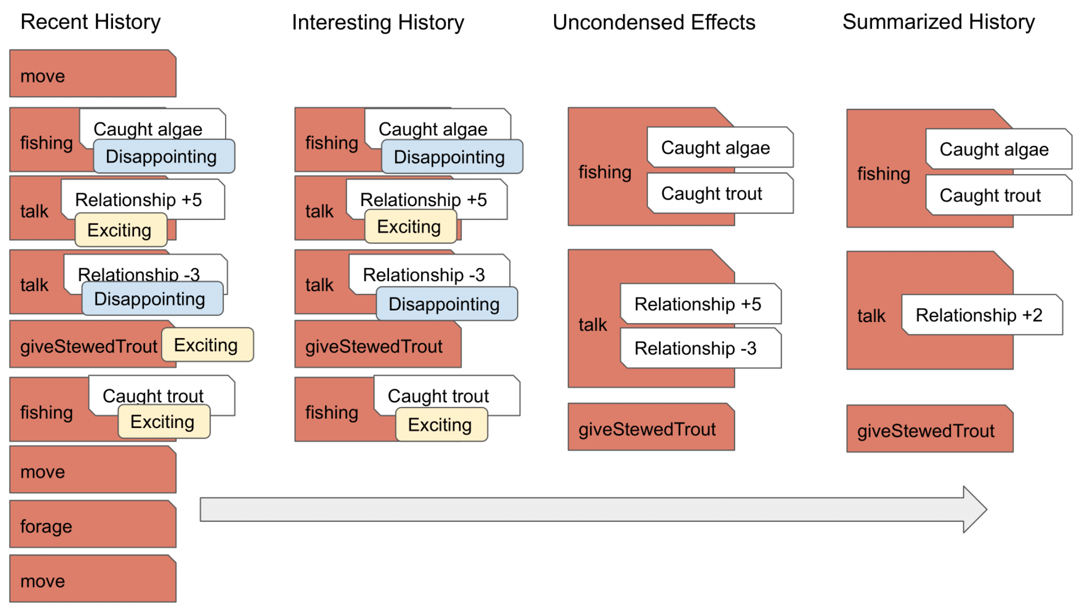


Figure 3.19: To filter interesting actions from the list of recent actions the NPC is aware of, we remove all "move" actions (actions in which the NPC moves from one location to another) and all actions not tagged as either "exciting" or "disappointing". Then it condenses similar actions (those with the same verb, actor, and feature) into a single action, combining all effects onto one action. Finally, similar effects are condensed into one effect. Notice that "Relationship +2" is not any of the original effects, but rather the sum of "Relationship +5" and "Relationship -3".

character fished at the forest river five times in a single day, all five of those actions, and the results of those actions were compressed into a single fishing action. This choice came with a minor drawback, in that even if an NPC performed an action a couple times in the morning and then a couple times in the evening it would be reported as if they did all of it at once. For the verbalization patterns we implemented (and will be discussing shortly), this was fine, but narratively is perhaps disappointing.

Lastly, we filtered actions according to whether the NPC reporting them thought the action was "interesting". An NPC could consider an action interesting for several reasons.

- If the action results, evaluated according to the Action Heuristic Manager, were different than the projected results.
- If the action resulted in an emotional status (described in Section [System Architecture]) in the NPC.
- If the action was taken to progress any player-directed goal.

Additionally, this opinion about an event was further refined into being "exciting" (the results were better than the projected results or the emotional status added was "happy") and "disappointing" (the results were worse than the projected results or the emotional status added was "sad" or "angry").

In doing this, we are able to concisely answer if "Anything interesting happen[ed] today?" And with the "excited" and "disappointed" variations, we can further hone in on questions such as "What happened today that's got you in such a good mood?" and "What's got you in such a bad mood today?"

3.2.2.2 Barkeep Engine

The single biggest difference between the conversation engine for the player and the conversation engine for the patron is that the one for the player usually returns many social moves instead of just one. However, the broad strokes for how it does this is still the same. Social moves are generated in response to prompts, and the general class of the response is based strongly on the general class of the prompt. For example,

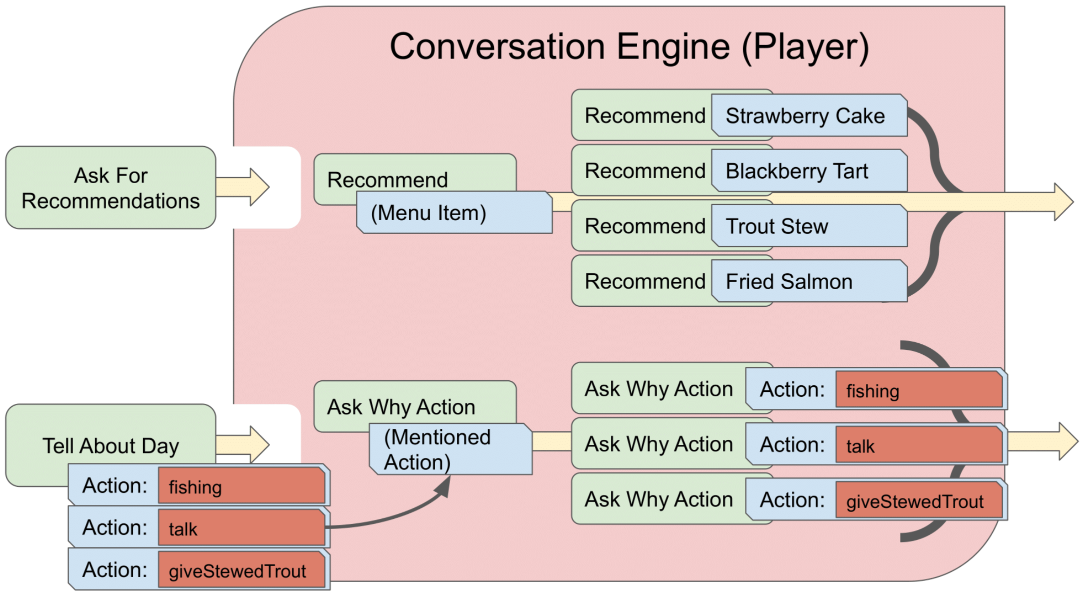


Figure 3.20: Unlike the version for the Patron, the Conversation Engine for the player returns sets of social moves which the player can choose from. Some NPC prompts result in the same set of player choices, as in the case of "askForRecomendation" which always results in the player being able to recommend one of the four things off of the menu. Other responses sets are generated off of the mentioned facts included in the NPC's prompt, as in the case of "tellAboutDay" where a "askWhyAction" social move is generated for each action in the mentionedFacts list of the prompt.

an "askForRecomendation" prompt is going to generate a set of "recommend" social moves, each with a different food item attached. A "tellAboutDay" prompt is going to generate a set of "askWhyAction", each with a different executed action included.

The second biggest difference between the Barkeep Engine and the Patron Engine is that not every prompt has a unique response in the Barkeep Engine. Rather, a great many things that the NPC can say do not warrant a direct response. For these kinds of prompts, the player is instead given access to a "main dialogue menu" which has access to a large collection of questions or statements to use to direct conversation.

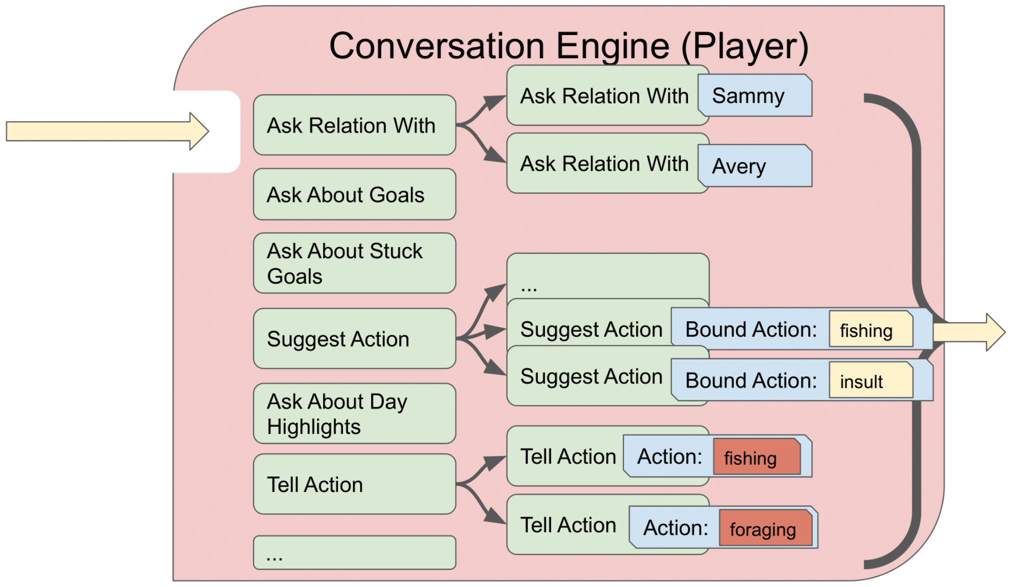


Figure 3.21: The player is also able to say things unprompted if the last thing the NPC said does not have a particular response. These social moves make up the "main dialogue menu". These include moves such as "askAboutDayHighlights", "tellAction", "askRelationWith", and "suggestAction". Some of these are complete statements in and of themselves, like "askAboutDayHighlights". Others are generated from the player's modeled knowledge, as in the case of "tellAction" which generates a "tellAction" social move for every executed action the player knows about that the NPC they are speaking to was not the actor or feature in. These many options are organized into submenus in an attempt to minimize overwhelming the player.

These include static social moves such as "askAboutGoals" or "askAboutDayHighlights" as well as social moves generated off of the player's knowledge base such as a "tellAction" for each executed action the player has learned about.

3.2.3 Verbalization

Social moves abstractly describe the turns of the player-NPC conversation, and the system has no problem deriving meaning from them or determining how to respond next. However, a human player would have a hard time gleaning any information

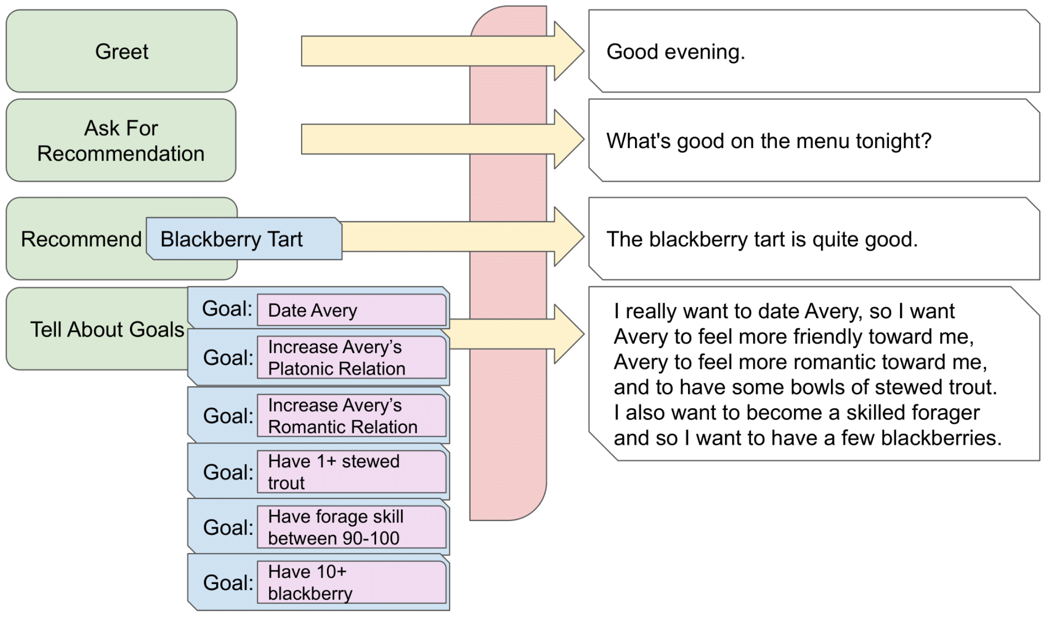

Greet
Ask For Recommendation
Recommend
Blackberry Tart
Tell About Goals
Goal: Date Avery
Goal: Increase Avery's Platonic Relation
Goal: Increase Avery's Romantic Relation
Goal: Have 1+ stewed trout
Goal: Have forage skill between 90-100
Goal: Have 10+ blackberry
Good evening.
What's good on the menu tonight?
The blackberry tart is quite good.
I really want to date Avery, so I want Avery to feel more friendly toward me, Avery to feel more romantic toward me, and to have some bowls of stewed trout. I also want to become a skilled forager and so I want to have a few blackberries.

Figure 3.22: Some examples of social moves verbalized into English text to display to the player.

from the raw system representation of social moves. This is where the Social Move Verbalizer comes into play. Its entire job is turning social moves into English, player-readable, dialogue. See Figure 3.22 for some examples of social move input and their verbalization output.

The first step in turning social moves into text is to pick an appropriate top-level pattern. Each social move has its own top-level pattern. Theoretically, it would have been possible to increase the variability of dialogue by providing multiple patterns for each social move to select from, but due to development constraints, this demo only implements one per social move.

Some of these are simple, and require no further processing, such as the "greet" social move, which is verbalized as "Good evening." or "askAboutState" which becomes

"How are you doing?" There are a number of other short acknowledgments and broad questions which fall into this category.

But for social facts which have parameters, the top-level pattern makes use of lower-level patterns. There are roughly three levels of verbalization. At the top level is the pattern dictated by the social move. This top-level pattern frequently makes use of mid-level patterns used to verbalize facts attached to the social move (Figures 3.23 and 3.24). Recall (as discussed in greater detail in Section 3.2.1) that many social moves carry references to facts that are spread between the player and NPC during conversation. Each kind of fact has at least one way in which it is verbalized, and most have a pattern for past, present, and future tense.

Within these patterns, at the lowest level, we additionally need to verbalize the names of objects and entities. Most objects have a system id that is different from the textual representation ("fishing_rod" versus "fishing rod" or "river_field" versus "the river running through the meadow" for example) and these need to be replaced as well. Additionally, most objects have a singular, plural, and generic form which are used in different ways throughout verbalization. For example, one could have "a bowl of stewed trout" (singular) or "many bowls of stewed trout" (plural) or one might just want to say they really like "stewed trout" (generic).

Complicating things further, we need to handle pronouns. All three characters use they/them pronouns in large part so we would not have to handle "she" versus "he" pronouns in verbalization. However, even with this, we still could not avoid needing to handle "I", "me", and "you" in verbalization depending on who was speaking to whom

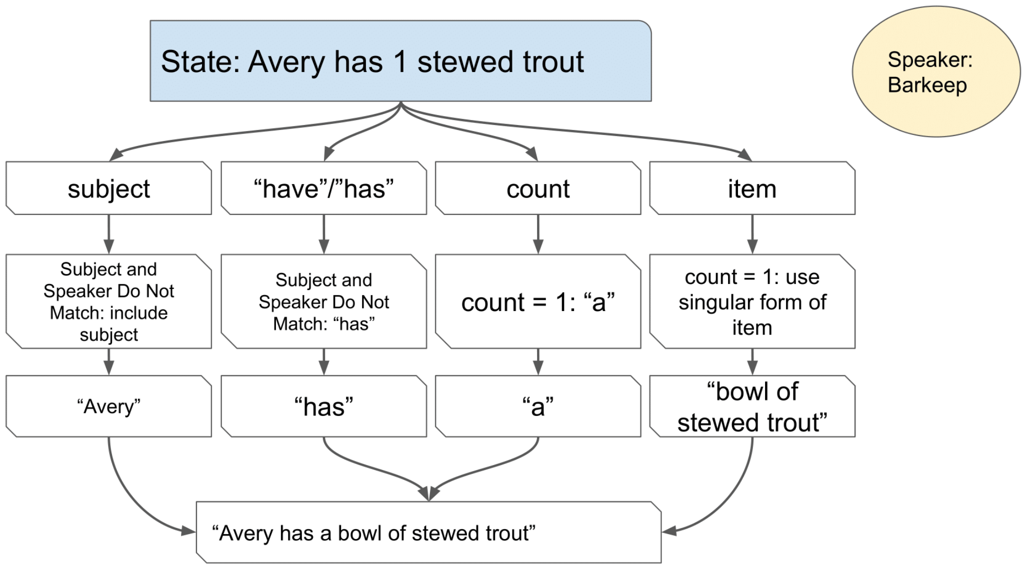

State: Avery has 1 stewed trout
Speaker: Barkeep
subject
“have”/”has”
count
item
Subject and Speaker Do Not Match: include subject
Subject and Speaker Do Not Match: “has”
count = 1: “a”
count = 1: use singular form of item
“Avery”
“has”
“a”
“bowl of stewed trout”
“Avery has a bowl of stewed trout”

Figure 3.23: To verbalize a fact about the current state of the world (in this case a fact about the state of Avery’s inventory), we first consider the general pattern for verbalizing inventory state: “[subject] [have/has] [count] [item]”. We then start filling in each of the fields. The subject is Avery. Grammatically, we want “has” since Avery is not the one speaking. Count could be filled in with phrases such as “some”, “a lot of”, “a few”, or “a” depending on how many items the state is describing. As there is only one, we will use “a” here. Similarly, because we are only discussing 1 item, we will use the singular form of the item “bowl of stewed trout”.

and who the subject of the conversation was.

To illustrate this process, consider the social move “tellAboutDay” (Figure 3.25). The “tellAboutDay” social move’s top-level pattern can be described as: “Today, [list of mentioned facts]”, where the “list of mentioned facts” are the facts attached to the social move in question. As such, to verbalize a “tellAboutDayEvents” social move, we first need to verbalize and then list those. For a “tellAboutDay” social move, these facts should be the executed actions that the speaking NPC considered interesting from the last 24 hours (see Section 3.2.2.1 for more details on how these were derived).

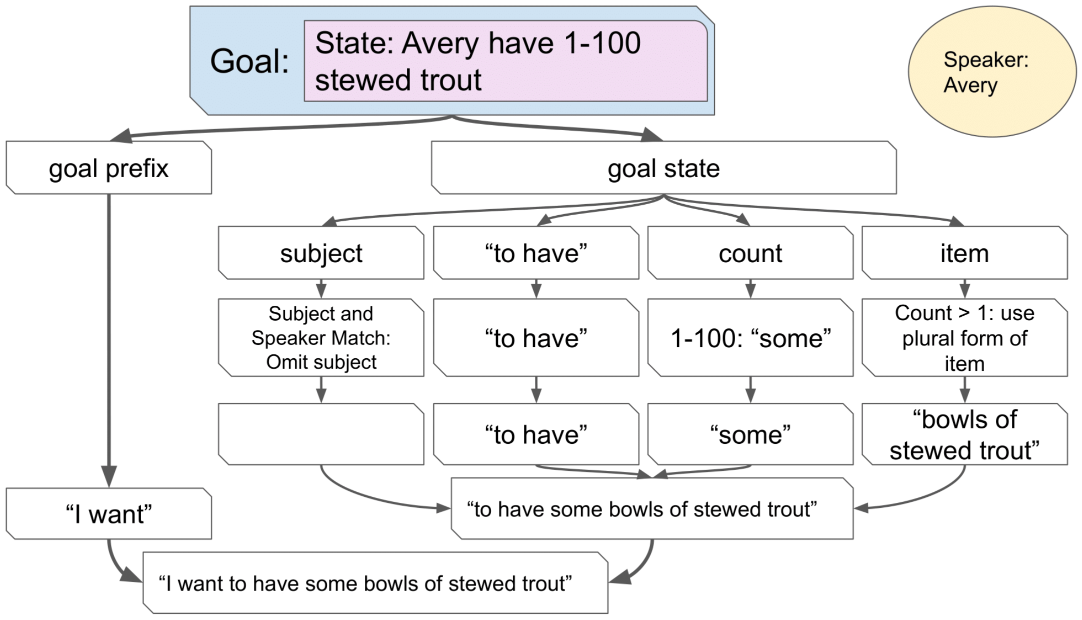

Goal:
State: Avery have 1-100 stewed trout
Speaker: Avery
goal prefix
goal state
subject
“to have”
count
item
Subject and Speaker Match: Omit subject
“to have”
1-100: “some”
Count > 1: use plural form of item
“to have”
“some”
“bowls of stewed trout”
“I want”
“to have some bowls of stewed trout”
“I want to have some bowls of stewed trout”

Figure 3.24: To verbalize a state when it is used as a goal state, we this time use the state’s goal form: “[subject] to have [count] [item]”. This will be included in a larger goal pattern specified by the social move this is attached to. (For the purpose of this example, it is “[subject] want [goal state]”).

We have discussed the contents of simulated actions previously in the World Simulation Section, but at that time, we did not describe the additional information that abstract actions carry about how to verbalize them. This additional data is necessary because there is no standard way to describe actions in the English language. To see why this is necessary, consider how one might say “I went fishing at the river in the forest” versus “I gave Avery a bowl of stewed trout”. At a glance, they have similar structures, both begin with the subject (“I” in both cases), are followed by a verb (“went fishing” and “gave” respectively), and then a feature (“the river in the forest” and “Avery”). However, notice there is actually another word in there between the verb and feature in the first sentence (“at”) which does not appear in the second. Perhaps we could include

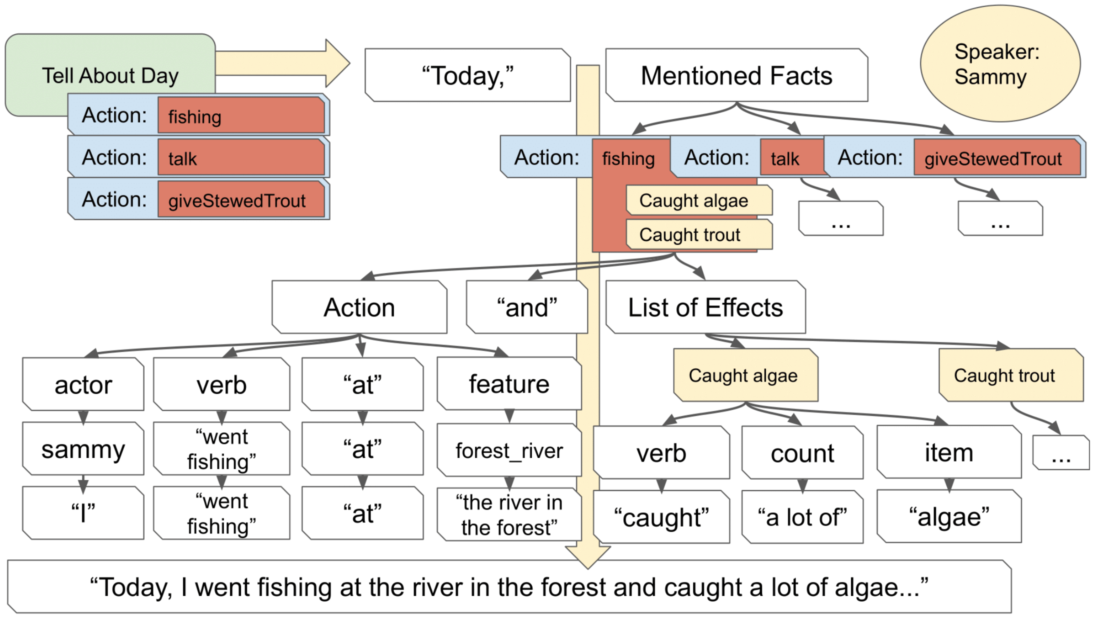


Figure 3.25: Verbalization of the social move "tellAboutDay" begins with the top-level pattern "Today, [mentioned facts]." Then we verbalize each fact attached to the social move. To do that, each action and all its effects are verbalized. Any nouns are turned into pronouns as makes grammatical sense according to who the speaker and the listener are and items are pluralized as makes numerical sense.

"at" in the feature name? Maybe. But only because there are no other actions that one can take on a river in this version of the demo. But what if there were? Could we guarantee every action taken on the river would appreciate the inclusion of "at" in the feature's name? And what about everything after the feature name? In the second sentence, it is a description of an item that was used in the action, but there is no analogous item to be included in the first sentence.

Worse, compare "I gave Avery a bowl of stewed trout" to "I asked Avery for a bowl of stewed trout". Again, these look like they follow the same pattern. They are both sentences about the transfer of items from one person to another. Both start with a subject ("I"), continue with a verb ("gave" and "asked"), followed by a feature ("Avery"), and then end with the item ("a bowl of stewed trout"). But once again, we trip over a small preposition right before the end, in this case "for". We can't just include that in the feature name this time either; there are too many other use cases for NPC names to even begin considering that as a possible solution. Similarly, it makes no sense to include it in the item name. As such, specifying how each action should be verbalized is the solution to the complexities of the English language that we chose.

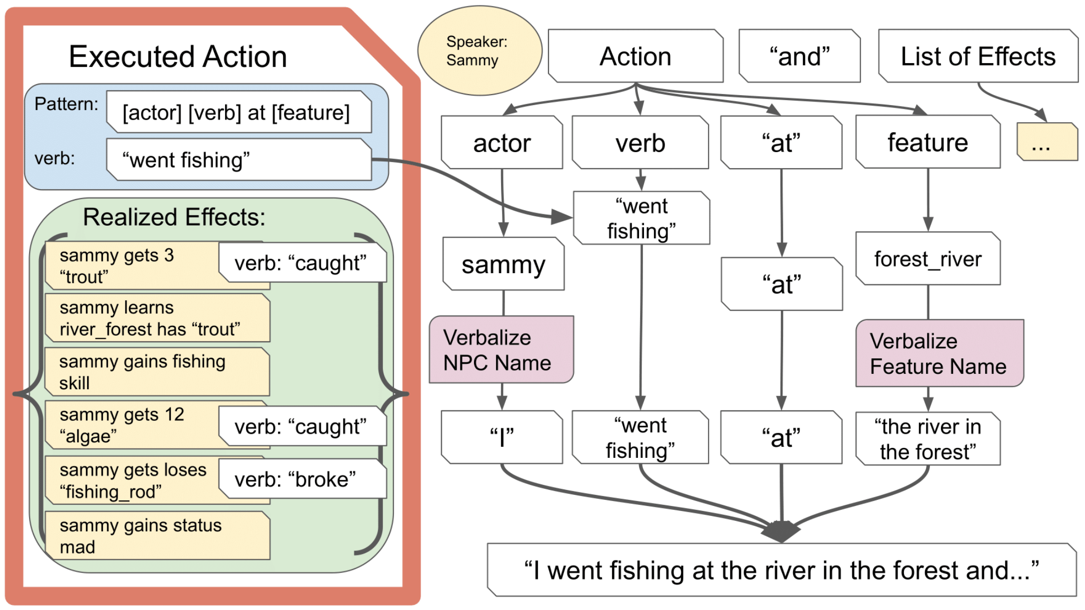


Figure 3.26: Here we can see that the executed action carries its own pattern for how it should be verbalized ("[actor] [verb] at [feature]") and the verb that should be used to describe it ("went fishing"). It should be noted that this verbalization verb ("went fishing") is different from the verb id mentioned in earlier descriptions of simulated action and used to identify the class of action in the world simulation ("fishing"). Each individual part of the pattern is evaluated and verbalized accordingly (NPC names are capitalized or replaced with pronouns. Feature ids are expanded into English descriptions.) and then combined according to the pattern.

In addition to describing the action itself, we want to describe the results of these actions. This is necessary as the outcome of an action is where the drama or information usually is. It isn't enough to hear how Avery complimented Sammy. We also want to know if they assumed Avery was lying to them or if they blushed bright red. In hearing that Finley went foraging, we want to know that they found strawberries there so we can tell Avery or Sammy where to find them.

How the results of actions should be verbalized is carried by the action's effects just like how the action carries its own verbalization data (Figure 3.27). For each effect, we can specify either a verb to describe the collection or loss of items or can write a short dialogue quip to describe the effect. Just like with actions, this allows greater flexibility when verbalizing effects. We can even leave this field empty, signifying to the system that we don't want the NPC to mention that particular effect in verbalization. The rationale here is that some results are implicit and don't need to be verbally commented on by the NPC (ex: improving as you perform an action; the NPC's emotional state as already reflected in the character portrait).

For example, in describing collecting trout, we can specify that the NPC should say they "caught" trout, while in describing collecting berries they should say they "found" them. For relationship changes, most actions that increase a relationship also have a potential outcome that damages the relationship instead. This makes things more complicated. Remember, as discussed in Section 3.2.2.1 (Patron Engine), the executed actions included in "tellAboutDay" are summaries of multiple similar actions, condensed down into a single event. Over the course of a day, it is possible for an NPC to talk to another NPC multiple times and over those several conversations get both

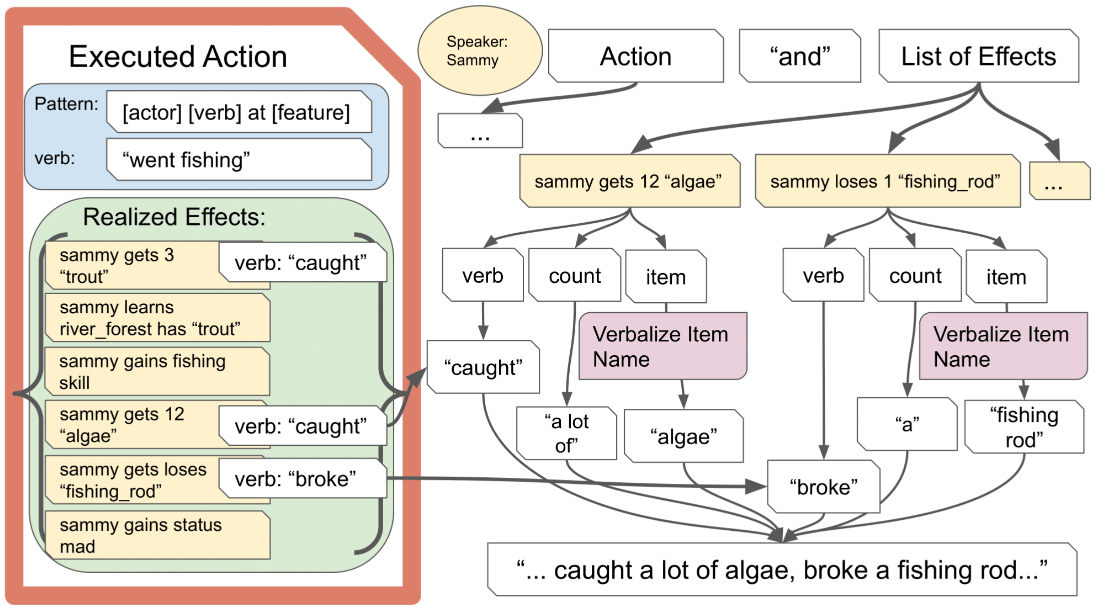


Figure 3.27: Like the action itself, each effect carries information on how it should be verbalized if at all. Here, only three of the six effects should be verbalized and all three use the same pattern. However, two different verbs are used in those patterns.

positive and negative relationship changes. When these effects are summed together, these values are also combined. So although Sammy might have talked to Avery three times, the first time increasing their friendship by five, the second time decreasing it by three, and the third time increasing it by another two, the executed action reported in "tellAboutDay" would have just one effect: that their relationship increased by four.

But, what verbalization should be used for this effect? The effect that increases friendship wants to be verbalized as "we had a good time" while the effect that decreases friendship wants to be verbalized as "it was frustrating". It would be strange to have an NPC report that they "talked to Avery and we had a good time and it was really frustrating." But how should the system know which of these to use? The answer is to have both of these verbalizations check what the value is and only display if it is above or below a certain threshold. The system allows any threshold to be used to switch between verbalization strings, but in this demo zero is the most common.

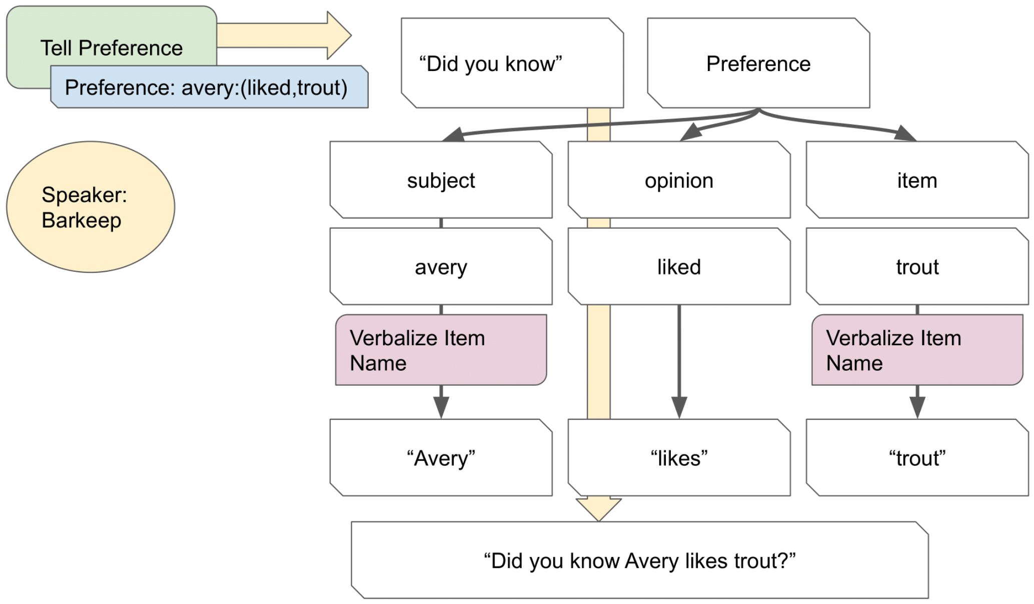


Figure 3.28: In the verbalization of item preferences, NPC names and item ids are verbalized just as they have in past examples. Preferences have five categories: loved, liked, neutral, disliked, hated. Each has an appropriate string to describe that system value which is used in these verbalizations. Just as in previous examples, each piece is evaluated and then combined.

Executed actions are not the only fact which is verbalized in this system. We are also able to verbalize things like NPC goals (Figure 3.24), NPC item preferences (Figure 3.28), and NPC relationships (Figure 3.29). Please see the related figure for examples on how these are verbalized.

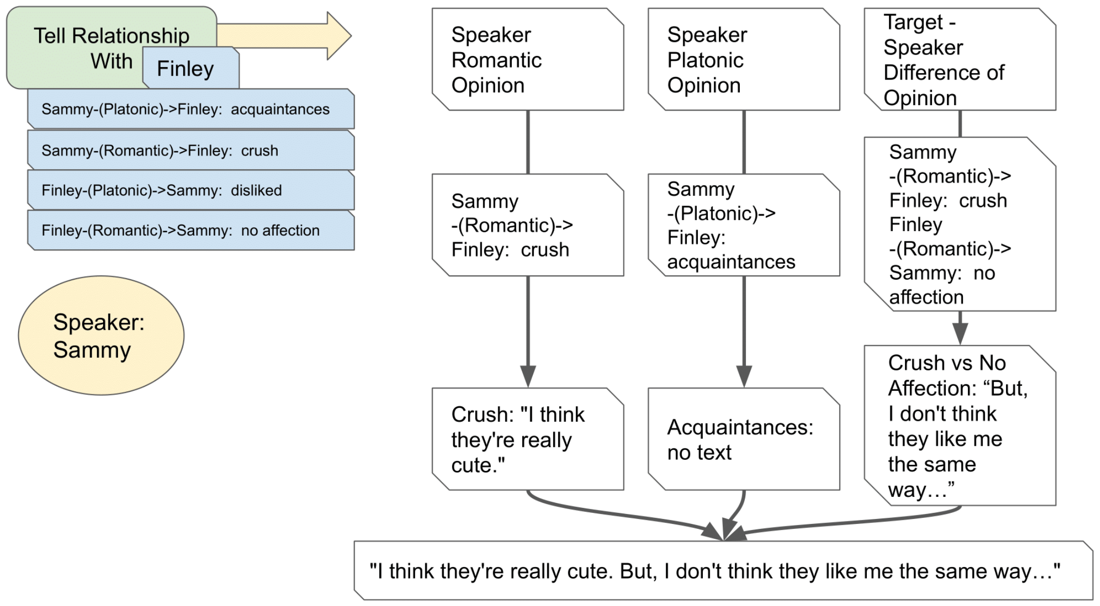

Tell Relationship With
Finley
Sammy-(Platonic)->Finley: acquaintances
Sammy-(Romantic)->Finley: crush
Finley-(Platonic)->Sammy: disliked
Finley-(Romantic)->Sammy: no affection
Speaker: Sammy
Speaker Romantic Opinion
Speaker Platonic Opinion
Target - Speaker Difference of Opinion
Sammy -(Romantic)-> Finley: crush
Sammy -(Platonic)-> Finley: acquaintances
Sammy -(Romantic)-> Finley: crush Finley -(Romantic)-> Sammy: no affection
Crush: "I think they're really cute."
Acquaintances: no text
Crush vs No Affection: "But, I don't think they like me the same way…"
"I think they're really cute. But, I don't think they like me the same way…"

Figure 3.29: To verbalize the relationships between NPCs, strings describing the broad potential relationships on each axis are combined to make one statement. There is an authored string for: romantically uninterested, romantic crushes, love, deeply in love, acquaintances, weak positive/negative platonic opinions, friends, best friends, enemies, and mortal enemies. Additionally, there are a set of strings to describe the difference of opinion between what the speaker feels about the subject and what they believe the subject thinks of them.

3.2.4 Interaction Between World Simulation and Conversation Space

Up to this point, we have largely just been talking about how actions taken in the world simulation populate the dialogue of the NPC in the conversation space. These actions, and the facts associated with them, are added to the barkeeper's knowledge base and made available for the player to share with other NPCs. As part of the NPC's decision-making process depends on their knowledge of world state (ex: where items can be found, opinions other NPCs hold of them, or preferences other NPCs hold toward items), players can indirectly alter what actions the NPCs choose to take.

For example, the player may want to pair up Sammy and Avery romantically. Since Avery already likes Sammy, the player's focus should be on helping Avery get Sammy to like them more. One method to do this would be for Avery to give Sammy their favorite food: strawberry cake. Sammy is not at all shy about telling the barkeep that they love strawberry cake, so it quickly becomes an option to tell Avery. Once Avery knows that Sammy loves strawberry cake, their Goal Manager is able to unroll Avery's "I want to date Sammy" goal into the desire to "give Sammy a strawberry cake". To give a strawberry cake, though, Avery needs to first have a strawberry cake. In this way player provided information propagates into new goals which result in different actions.

Additionally, the player can suggest actions for the NPCs to perform directly with the "suggest" social move. When they suggest an action, performing that action becomes a high-priority goal for the NPC to accomplish. Players can use this for good or for ill, for example suggesting characters "ask out" others, either because the player knows they both secretly love each other and they just need that last push, or because they wish to sour relations between two characters that don't feel that way toward each other.

The "suggest" social move can also be used to tell characters about actions the NPCs did not know were options in the world simulation. For example, there are four cooking actions each of which cooks a different dish. The NPCs do not know they can perform these actions unless the player suggests they try cooking that dish.

Between subtly telling NPCs new information to change their understanding of world state and directly giving NPCs directions to follow, players theoretically have significant control over the system. We shall see if players felt that or not in the following section.

Chapter 4

Playtesting

In evaluating Crosston Tavern, we had seven participants do initial playtesting on the game demo. Players were first asked about their experience with video games. This included what genres of games they typically play and if they had any experience with specifically Farming Sims (such as Harvest Moon or Stardew Valley), Animal Crossing, or the Sims. These were selected to be of particular interest as Crosston Tavern is meant to be an experience in the same larger genre of casual social game as these three.

Participants were then asked to play the game for at least twenty minutes but were allowed to continue for as long as they wanted up to fifty minutes. They were asked to think aloud as they played, and we took notes on their reactions. It was suggested that players try to manipulate the relationships between the NPCs.

After participants were finished, we asked a series of follow up questions:

- Did you enjoy yourself?

- Did you find what the NPCs had to say interesting?
- Did you feel you were successful at figuring out what each character liked?
- Did you feel you were able to manipulate their relationships?
- Was there anything you wanted to ask the NPCs or say to them that you were unable to?
- Did you find it easy or difficult to say what you wanted to the NPCs?
- Were there any particular moments that stick out to you from your playthrough?
- Do you have a favorite NPC?
- Would you have liked to talk to more townies, or was it too much juggling three as it was?
- Would you have liked to hear about the actions of additional background characters who did not show up in the bar?

Overall, the reception of Crosston Tavern was positive, with players saying they enjoyed playing it. The average playtime was about half an hour, with the longest playtime forcefully ended at fifty minutes. Two players stated they wanted to play it more after the playtest was complete.

Most playtesters reported finding the dialogue initially interesting but that they quickly found the way in which the NPCs talked repetitive. One playtester enjoyed the repetitiveness of the dialogue, saying it was “nonsensical” but that “it was natural to the setting". Of the things the NPCs said, most playtesters reported being more interested in dialogue about inter-character relationships and were less interested in statements about the physical state of the world or resource collection.

Most players reported picking up the item preferences of the NPCs through gameplay. Observations during play support this, as players frequently served NPCs their favorite foods after one or two in-game nights. Players largely did not experiment with other menu items once they had determined which food items a character liked.

It was suggested that players attempt to manipulate the relationships between NPCs. Players showed moderate to high investment in that player goal. Most frequently players tried to romantically pair up characters. There was an even distribution of people wanting to pair each of the three potential pairings (Avery and Finley, Finley and Sammy, Avery and Sammy). A few players changed their minds about who they wanted to help get together over the course of play as they learned more about the character's interactions. For example, one player had initially wanted to pair Finley and Sammy, but after hearing Finley had turned Sammy down and then insulted them they switched their goals to supporting Sammy and Avery.

A smaller number of players wanted to see all three characters get along as friends (or at least actively did not want them to be enemies), although only one player said this was their primary goal.

One player, partway through play, decided to become "an agent of chaos" with the explicit goal of causing drama and driving wedges between the characters. This particular player expressed an almost guilty pleasure in purposefully suggesting actions to the NPCs which would sour their relationships, such as suggesting they insult one another. Frequently they said things such as, “Oh, god, I feel so bad. I am a terrible person” while grinning ear to ear at the chaos their most recent action had sown.

Another player expressed a similar sentiment. They described the feeling of playing Crosston Tavern that of being responsible for the lives of the NPCs. That “there is guilt attached to messing up but also delight” and that they liked “the hubris in the responsibility because the animals do whatever you tell them to.”

Unfortunately, not every playtester felt that their actions strongly affected the NPCs. Two of the seven playtesters reported feeling like they were unable to manipulate the world state, while another three described it as feeling like their ability to change things was moderate or would be more pronounced given a longer playtime. The two who felt most strongly that their actions affected the system were those who had (accidentally or intentionally) ruined a relationship between two characters.

Overall, players were reasonably satisfied with the range of things they could say to the NPCs. A few players wished that there were more dialogue options relating to character relationships, for example, they wanted to tell Sammy about Finley’s opinion of Avery, while the current system only allowed the player to tell Sammy what Finley might think of Sammy or what Avery might think of Sammy.

Additionally, players had some difficulty navigating the existing dialogue options menu, especially early in play. Most players were able to find what they were looking for without too much difficulty by the end, but a few were still searching through the menus for the one dialogue option, in particular, they were looking for. A frequent point of confusion was with the “Did you know. . . ?” interaction. Several players at several points within a playthrough thought that this was the heading for telling characters about other character relations (statements of the form: “Did you know so-and-so likes so-and-so?”) when instead this was the heading for telling characters about other character’s item preferences (statements such as: “Did you know so-and-so likes strawberries?”).

When asked if there were any moments that stood out to players, most replied with an event centered on the NPC’s emotions. Examples include Finley entering the bar frustrated three days in a row, Sammy crying as they told the player about being rejected in asking out Finley, and characters blushing after being told their crush likes them. Players also strongly remembered events which they felt responsible for or which they felt some guilt about, such as when they lied to one of the characters about how another NPC felt about them or when they heard their suggested action backfired.

When asked which of the NPCs was their favorite, Sammy was selected the most, while Finley and Avery were named with equal frequency. Two players stated it was because they thought their pick was the cutest of the three, one specified it was specifically because of the character’s antics, another three players explained that their pick was the sweetest or most interesting. We find it particularly interesting that some players described certain NPCs as “sweet” or “interesting” as all three have the same decision-making processes — only their goals differ. It would be interesting to see if switching the character sprites would significantly change player’s perceptions of the three NPCs or if these perceptions would remain tied to the particular goal sets.

When asked how players felt about the size of the cast, opinions were largely unanimous. All players agreed that three cast members were the minimum that sounded interesting. This makes sense as all players showed the most interest in inter-NPC relations and reducing the cast size from three to two greatly reduces the potential character dynamics. Some players suggested that perhaps starting with a smaller number of NPCs per night, then having three from a larger cast as the nights progress might be interesting as well. Additionally, there was overwhelming interest in having a cast of background characters who do not appear in the bar but which the bar patrons talk about. This was part of the original plan but was cut because of time constraints.

Overall, most players were most interested in the social relationships between NPCs as well as events that triggered an emotional response in those NPCs while dialogue related to resource gathering was considered significantly less interesting or noteworthy. Interestingly, however, several players mentioned liking that NPCs had non-relationship goals, even though those goals largely had to do with resource-gathering actions. One player mentioned that "it was cute that the characters all had their own goals" and that it was "almost more interesting than the dating stuff". Additionally, another player thought that a character (Avery) was "pushy" because dating another character (Sammy) seemed to be their number one priority, but preferred another character (Sammy) because they felt that they were more interested in their profession goal. This is especially interesting because all three characters begin the simulation with similar goal states.

Chapter 5

Future Work

Crosston Tavern successfully demos a new character interaction mechanic that combines planning-based simulated characters with conversational interaction. In this section, we discuss both game features that would transform the demo into a fully-produced game, as well as future directions for the AI framework.

The Crosston Tavern that exists currently features three townies who visit a Tavern every evening. These are the only entities that exist in this version of the world. However, in the initial design, they were to be joined by a host of background characters who would additionally take action in the world and whose actions could be observed and reported on by the primary cast. For the purposes of this demo these characters were never developed, in part due to the time that would have been required to design appropriate goals for each, and in part, because it was nontrivial to design appropriate metrics for when the main cast should report the actions of others. However, when discussing this dropped feature with the playtesters, we were met with an overwhelming interest in the feature.

Additionally, the actions available to our cast of NPCs are fairly limited. The space of goals available to them and the range of options they may take to achieve those goals is also more limited than was initially desired. Significantly more time could be spent engineering additional actions as well as world state to support more complicated and more varied goals.

For example, early designs included characters who are potentially unemployed and looking for work. Jobs would become available if an NPC with a job found themselves very busy and in need of help. However, missing from the current design of the world state is a concept of being busy. This could perhaps be implemented by having the NPC notice when a profession-related goal is not being advanced over a given time period triggering a goal to hire help. This is an example of the richer interactions that would be enabled with a more complex goal and action collection, as well as additional reasoning about goals.

Another direction to explore would be to increase the dynamics of the simulated world. In a fully-featured farming sim world, the world would change with the seasons, with each season making available different fish, foragables, and crops. This would require a more complicated understanding of world state on the part of the NPCs, however. At the very least, one would need to increase the number of resource categories on these features to have a resource list for each season and add additional cases to the list of potential outcomes for actions that are affected by this kind of seasonal change. Alternatively, it might be more elegant to have a seasonal variant for each feature with their own resource lists and systematically swap them out as the seasons change. In either case, we would likely want NPCs to be able to use the seasonal variants in their reasoning about what they might get from the current season as these lists in farming sims frequently feature significant overlap. This kind of reasoning would likely want to make use of some sort of "confidence" in the NPCs beliefs, much like *Talk of the Town* [23] NPCs do.

This would have the added benefit of allowing NPCs to reason over things that inherently change, such as the goals of other NPCs (something they do not currently reason about), and might more naturally allow uncertainty in the state of their relationships (where currently they can be wrong but will still wholeheartedly believe that incorrect opinion).

Additionally, this might open up the possibility of NPCs lying to each other and the player about the state of their relationships or their desired goals. Alternatively, it might allow NPCs to not immediately completely believe what the barkeep tells them. How this might change play would be interesting to experiment with.

Work could also be done to allow NPCs to talk to each other in the conversational space. How three-way conversations might play out could result in some interesting conversations. Since NPCs and players both use a conversation engine to pick their next social move this is not outside the realm of possibility for this system. However, significant logic would need to be added around picking who should respond to the current speaker. Additionally, questions of whether, and under what conditions, a speaker can direct statements to a specific conversational partner would need to be addressed. Additionally, we would need to increase the social moves available to NPCs. In the current demo, NPCs do not have access to most of the probing questions the player can use (moves like “askAboutDayHighlights”, “askAboutGoals”, or “askRelationWith”) in large part because the player does not have answers to those kinds of questions. The barkeep has no simulated day to report to “askAboutDayHighlights”, they have no system-recognized goals to tell, and no formal relationships with the NPCs. But, if NPCs could talk to each other, it would be reasonable for them to ask these things of each other.

Additionally, there was the intent that the NPCs would be able to direct the conversation to a greater degree than they currently do. This could be done through an Ensemble-Engine-style reasoning system, with NPCs picking their next social move by evaluating the social situation against a set of influence rules. How NPC agency would be balanced against player agency is a question that would be interesting to explore in such a system.

Future work might also look into how else to narrativize simulated action. In the current version, characters summarize actions of the same type taken on the same feature into a single event for the sake of reporting to the player. However, one could imagine more complicated patterns of actions being summarized in specific ways. For example, several actions all taken in the forest, at least one of which being a social action with another character might get summarized as “me and Sammy met up in the forest and fished a bunch together.” The work of Max Kreminski [16]’s story sifting might have interesting applications here.

On the conversation side, there is no question that more varied and more grammatically correct dialogue patterns would improve the player experience. In the current demo, each social move has one verbalization pattern. Some of these were noticed very quickly by players, to their displeasure. For example, the tendency for NPCs to tell the player every evening in the same way about what they think of the food they ordered (an order which is very frequently the same thing they ordered the night before and the night before that) was quickly noticed and reported as mildly annoying. As this is part of the "thank" social move's verbalization pattern, this annoyance might have been mitigated if there had been multiple patterns the NPCs could use to express this sentiment. Additionally, NPCs generally have only one possible response to any given player input. Giving them more social moves that could be used in response to player dialogue would also mitigate the problem of repetition.

Additionally, despite our best efforts, there are a number of places where the verbalization patterns result in grammatically incorrect phrases or phrases that, though technically grammatically correct, are strange to an English-speaking ear. For example, it is possible for the barkeep to tell an NPC about another pair having a good time talking. This gets verbalized as something along the lines of "Did you hear Sammy talked to Finley and we had a good time?" This happens because "we had a good time" is a hardcoded phrase that is used to signify that the total relationship change was positive (See Section 3.2.3 [Verbalization] for more details on how actions and their effects are generally verbalized). To fix this, we would need to implement a larger set of verbalization patterns and support the system's use of the pronoun "we".

This system, or a system based on this system, might be applied to farming sims. The simplest implementation would be to apply this same system to the inn/-tavern, but instead of being the barkeep talking to patrons, you play as a townsperson talking to their neighbors. It would give the player a reason to interact with the inn (something there is little reason to do in most farming sims despite the ubiquity of this building). Every evening the player could come and chat with other townspeople, learn the latest gossip, and influence the relationships between other NPCs (something generally not modeled in farming sims). Perhaps clearing up old rivalries could be necessary to advancing the plot or unlocking certain services (like the carpenter won't build barns on principle until the rancher apologies to them for some past slight).

A similar sort of scheme might be applied to RPGs, particularly those with a central hub area the player repeatedly and frequently returns to. RPGs already frequently feature player-NPC interactions. This would be an expansion on that, increasing the ways to interact with both party members and townspeople. Like in regard to farming sims, improving relationships between NPCs might unlock new services or areas. Perhaps new armor or weapons could become available as craftsmen in the central hub area become more willing to work together. Perhaps a friendship built between an alchemist and a knight results in the pair exploring the wilds to find the alchemist an herb that results in a new dungeon being discovered.

Perhaps regardless of setting, players would enjoy romantically pairing up NPCs. It is already popular to woo NPCs yourself, with both most RPGs and farming sims allowing the player to marry an NPC. Why not share that joy by pairing up your favorite NPCs with each other?

We are currently considering a follow-up game using this system in which one plays not as a barkeep but as a Guild Master of an adventurer's guild, set in a fantasy RPG style world. The player would be responsible for helping NPCs form adventuring parties that work well together (both in terms of party composition and personality interactions) and finding them jobs of about their skill level, all with the hope that this batch of young adventurers listen to you and don't get themselves killed.

Chapter 6

Conclusion

This thesis is an exploration of player-NPC interaction and how that can be supported by simulated NPC action. In it, we have attempted to flip the standard roles of NPCs and players in video games. Where many past games have placed NPCs in the role of support of player goals, Crosston Tavern instead has the player support NPC goals. In this demo, the NPCs are free to pursue their personal goals within a simulated world as well as report on those actions and their understanding of the world to the player. The player uses this reporting to guide the NPCs in their efforts, either by sharing information or suggesting specific actions to take next.

Built on a simulation system, NPCs autonomously take actions toward their personal goals. Those goals are not set by the author but rather derived from their interpersonal relationships and their profession in the simulated world. They are additionally able to break down these top-level goals into intermediate sub-goals by reasoning over the space of possible actions and their understanding of world state. By evaluating their available actions against these goals, NPCs are able to take autonomous actions, in some cases even achieving their goals without the aid of players.

However, in many cases, NPCs do not have all the information they need to achieve their goals. In other cases, NPCs can be redirected to pursue other goals. Here is where the player exerts their influence over the system. By sharing stories told to them in the conversational space with other NPCs, the player can tell NPCs about the world they live in they previously did not know. They also might speculate about the feelings other NPCs hold for their conversation partner, or perhaps suggest new actions to try out. Any of these could be the push an NPC needs to achieve one of their long-standing goals or to shift their understanding of the world so their goals themselves change.

Initial playtests of this demo have been positive. Players are excited by the potential for drama this system does and could support. They suggest there is much interest in exploring inter-NPC relationships through gameplay and delight in seeing emotional reactions from player action. We look forward to seeing this approach to player-NPC interaction or inter-NPC relationships used in future contexts and hope this is only the beginning for this line of research and design.

Bibliography

[1] Amccus. Harvest moon. SNES, 1996.

[2] Marvelous AQL. Story of seasons: Pioneers of olive town. Nintendo Switch, 2021.

[3] Roberto Basili and Maria Teresa Pazienza. *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing: 10th Congress of the Italian Association for Artificial Intelligence, Rome, Italy, September 10-13, 2007. Proceedings. Roberto Basili, Maria Teresa.* Springer-Verlag Berlin Heidelberg, 2007.

[4] ConcernedApe. Stardew valley. `store.steampowered.com/app/413150/Stardew_Valley/`, 2016.

[5] Frontier Developments. Planet zoo. `store.steampowered.com/app/703080/Planet_Zoo/`, 2019.

[6] Quantic Dream. Detroit: Become human. `store.steampowered.com/app/1222140/Detroit_Become_Human/`, 2020.

[7] Mirjam Palosaari Eladhari. Ai based game design. Presented in GDC Education Summit, 2015.

[8] Double Eleven and Introversion Software. Prison architect. `store.steampowered.com/app/233450/Prison_Architect/`, 2015.

[9] Nintendo EPD. Animal crossing: New horizons. Nintendo Switch, 2020.

[10] R. Evans and Emily Short. Versu—a simulationist storytelling system. *IEEE Transactions on Computational Intelligence and AI in Games*, 6:113–130, 2014.

[11] The Pokemon Company Game Freak, Nintendo. Pokemon ruby, 2002.

[12] Supergiant Games. Hades. `store.steampowered.com/app/1145360/Hades/`, 2020.

[13] Telltale Games. The walking dead game. `store.steampowered.com/app/207610/The_Walking_Dead/`, 2012.

[14] Golden Glitch. Elsinore. `store.steampowered.com/app/512890/Elsinore/`, 2019.

[15] Marvelous Interactive. Harvest moon: Tree of tranquility, 2007.

[16] Max Kreminski, Melanie Dickinson, and Noah Wardrip-Fruin. Felt: a simple story sifter. In *International Conference on Interactive Digital Storytelling*, pages 267–281. Springer, 2019.

[17] Maxis. The sims 4. `www.ea.com/games/the-sims/the-sims-4`, 2014.

[18] Joshua McCoy, Mike Treanor, Ben Samuel, Aaron A. Reed, M. Mateas, and Noah

Wardrip-Fruin. Prom week: Designing past the game/story dilemma. In *FDG*, 2013.

[19] Joshua McCoy, Mike Treanor, Ben Samuel, Aaron A. Reed, M. Mateas, and Noah Wardrip-Fruin. Social story worlds with comme il faut. *IEEE Transactions on Computational Intelligence and AI in Games*, 6:97–112, 2014.

[20] James R Meehan. Tale-spin, an interactive program that writes stories. In *Ijcai*, volume 77, page 9198, 1977.

[21] Neverland. Rune factory 4. Nintendo 3DS, 2012.

[22] J. Ryan, A. Summerville, M. Mateas, and Noah Wardrip-Fruin. Toward characters who observe, tell, misremember, and lie. 2015.

[23] James Ryan. *Curating Simulated Storyworlds*. PhD thesis, UC Santa Cruz, 2018.

[24] Ben Samuel, Aaron A Reed, Paul Maddaloni, Michael Mateas, and Noah Wardrip-Fruin. The ensemble engine: Next-generation social physics. In *Proceedings of the Tenth International Conference on the Foundations of Digital Games (FDG 2015)*, pages 22–25, 2015.

[25] Ben Samuel, James Ryan, Adam J Summerville, Michael Mateas, and Noah Wardrip-Fruin. Bad news: An experiment in computationally assisted performance. In *International Conference on Interactive Digital Storytelling*, pages 108–120. Springer, 2016.

[26] Giants Software. Farming simulator 17. `store.steampowered.com/app/447020/Farming_Simulator_17/`, 2016.

[27] Bethesda Game Studios. The elder scrolls v: Skyrim. PC, PS3, Xbox 360, 2011.

[28] Bethesda Game Studios. Fallout 4. Windows, PlayStation 4, Xbox One, 2015.

[29] Digital Sun. Moonlighter. `store.steampowered.com/app/606150/Moonlighter/`, 2018.

[30] Mike Treanor, Alexander Zook, Mirjam Palosaari Eladhari, Julian Togelius, Gillian Smith, Michael Cook, Tommy Thompson, Brian Magerko, John Levine, and Adam Smith. Ai-based game design patterns. 06 2015.

[31] Valve. Portal. `store.steampowered.com/app/400/Portal/`, 2007.

[32] Valve. Portal 2. `store.steampowered.com/app/620/Portal_2/`, 2011.

[33] ZA/UM. Disco elysium. `store.steampowered.com/app/632470/Disco_Elysium__The_Final_Cut/`, 2019.