TÉCNICO LISBOA

NPCs based on Agency Models for Survival Games

Don’t Starve Together NPCs with FAtiMA

Fábio Vieira de Almeida

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Professor Rui Prada
Samuel Mascarenhas

May 2018

Agradecimentos

Primeiro que tudo, quero agradecer aos meus pais por todo o seu suporte e amor incondicional ao longo de todos estes anos. Vocês deram-me a oportunidade de seguir os meus sonhos. Por isso, e tudo o resto, muito obrigado.

Gostava também de agradecer aos meus orientadores, o Professor Rui Prada e o Samuel Mascarenhas. Foram vocês que me deixaram seguir as minhas ideias guiando-me da melhor maneira possível. Pelas inúmeras revisões, conselhos e paciência, o meu mais sincero obrigado.

Quero também deixar uma palavra de apreço à República ”A Desordem dos Engenheiros”. Mais do que uma casa, um verdadeiro lar onde passei todos estes anos. Foi aqui que me tornei quem sou e nunca esquecerei todos os momentos que aqui passei. A todos os que por cá passaram, estão a passar e passarão: ”**Somos filhos da Desordem** (...)”.

Ao Rui, ao Ricardo, ao Gonçalo, ao Sérgio, ao Diogo, ao Manuel, ao Pedro, ao Rafael, ao Henrique, à CPLEIC, ao Pedro, ao Ricardo e a todos os outros que por falta de espaço ou memória não estão nesta lista. Obrigado pela vossa amizade e companheirismo ao longo desta jornada que é a faculdade. Foi um prazer e um privilégio fazê-la convosco.

E finalmente, à Teresa. Por tudo o que aturas e me fazes aturar, por me apoiares e puxares por mim, e por me amares como te amo a ti.

Abstract

Today's video games are striving to maintain high levels of fidelity. The realistic graphical representation of virtual worlds is only betrayed by the lack of believability that in-game characters present. To maintain the immersion created by exquisite graphics, characters must be able to create the illusion of life, which requires them to possess basic human traits like social awareness, reactivity, and active goal pursuit. Some game genres, like role playing games, have seen this problem being addressed by using agency based characters. However, survival games have not been subject to the same attention. In this work, we address this issue by proposing a framework that allows developers to create characters based on agency models for survival games. By making use of FAtiMA Toolkit, a fully fledged model for agency, and Don't Starve Together, a popular survival game, we've implemented and published such a framework with an example character, Walter. Walter has been run and tested against a behaviour tree based agent.

Keywords

Artificial Intelligence; AI; NPC; Non Playable Characters; Believable Characters; Survival Games.

Resumo

Os videojogos de hoje em dia batalham por manter altos niveis de fidelidade. As representações altamente realistas dos mundos virtuais apenas são traídas pela falta de credibilidade dos personagens que os habitam. De modo a manter a imersão criada por gráficos extraordinários, os personagens têm de ser capazes de criar a ilusão de vida, o que requer que possuam características humanas como consciência social, reactividade e procura activa de objectivos. Alguns géneros de jogos, como RPGs, já viram este problema ser endereçado pela utilização de modelos sociais de agência. No entanto, videojogos de sobrevivência ainda não foram alvo deste tipo de atenção. Neste trabalho, endereçamos esta falha propondo um sistema que permite criar personagens credíveis para videojogos de sobrevivência. Usando o FAtiMA Toolkit, um modelo social para agência por direito, e o Don't Starve Together, um videojogo de sobrevivência popular, implementámos e publicamos o sistema proposto e um personagem de exemplo.

Palavras Chave

Inteligência Artificial; IA; Personagens Não Jogáveis; NPC; Personagens Credíveis; Videojogos de Sobrevivência;

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
ACL	Agent Communication Language
CiF	*Comme il Faut*
DLL	Dynamic-Link Library
DSL	Domain Specific Language
DST	Don't Starve Together
FAtiMA	Fearnot AffecTIve Mind Architecture
FIPA	Foundation for Intelligent Physical Agents
GUI	Graphical User Interface
GUID	Globally Unique Identifier
HTTP	Hypertext Transfer Protocol
IAT	Integrated Authoring Tool
JADE	Java Agent DEvelopment Framework
JSON	JavaScript Object Notation
MCTS	Monte Carlo Tree Search
NPC	Non Playable Character
RPC	Role Play Character
RPG	Role Playing Game
WFN	Well Formed Names

1

Introduction

Contents

In recent years, the continuous improvement and development in computer graphics has pushed video games to new levels of graphical fidelity. Graphical representations of virtual worlds have become increasingly more realistic, allowing players to experience new levels of immersion [1]. Additionally, with the recent boom in virtual reality systems, players have never been so physically immersed in a game's virtual world.

The increases on fidelity and real lifelikeness of the virtual world representation, cause players to create expectations on the interactions they can have on the world. This expectation applies not only to interactions with the virtual world itself, but is also extended to the interactions with the characters that compose the world, typically called Non Playable Characters (NPCs).

Computer and player controlled characters need to act in a believable manner so that the illusion of reality created by exquisite graphics and physics, the player's immersion, is not broken [2]. The lack of believability in NPCs can break the player's immersion and can provide a bad gameplay experience, given that the game's flow and player's immersion are key elements to the gameplay experience [3]. Reactivity, goals, emotions, and social competence are necessary traits NPCs need in order to be believable [4].

Some game genres, like Role Playing Games (RPGs), which are heavily dependent on player to NPC interaction, have received special attention on the creation of believable characters. However, other game genres, like survival games, have not received such attention.

Survival games, a genre that has seen a rise in popularity recently, are often based in open world scenarios that can be procedurally generated. Either single or multi-player, some survival games revolve around resource collecting, crafting, and management of the character's needs (hunger and thirst, mostly), while others are set in supernatural worlds with some elements of horror: monster infested worlds where players have to survive by defeating them.

More often than not, survival games lack a specific objective other than surviving in a harsh world for as long as possible. However, survival games are trending among players and game developers alike, seeing great attention in the indie game scene.

Don't Starve [5], for example, is a single player survival game set on a procedurally generated world where players have to collect, craft, explore, and fight to survive. With a day and night cycle, Don't Starve provides an interesting gameplay by combining mechanics for temperature, wetness, hunger, health, and sanity. The introduction of sanity to the game's character provides a novelty factor among its peers, giving the character a human-like trait appropriate for a survival game, representing the strain put on the character by having to survive the harshness of a world with nothing to start with.

Following the trend for multiplayer survival games, Don't Starve as also been released as a multiplayer game under the name of DST [6]. The game is identical to its predecessor, though it allows several players to join the same world. The multiplayer nature of the game brings new challenges to

players that can either cooperate or compete for survival. In extreme cases, players can even attack and kill each other giving the game an extra challenge to overcome.

However, the game counts with no NPCs to accompany players in their journey for survival, making players dependent on each other for gameplay. The few autonomous non-aggressive characters of Don't Starve Together present limited behaviour and are not subject to the same rules as the players (hunger, health, sanity, etc.).

The nonexistence of interesting and robust Artificial Intelligence (AI) for characters is not exclusive to Don't Starve Together. Other survival games suffer from the same problem: the lack of collaborative AI for NPCs capable of creating believable characters.

As said before, on other game genres like RPGs, tools exist and have been used for the creation of AI controlled NPCs that take into account the character's believability [7] [8] [9]. Usually empowered by social models, these tools have not yet been applied to survival games.

1.1 Motivation

In order to improve player's game play experience, the game industry has used AI with several different purposes: player experience modeling, procedural content generation, massive-scale game data mining, and NPC AI [10].

Many modern day video games are dependent on player to NPC interaction, and while some degree of independent decision making is implemented through the use of AI techniques, it's mostly based on combat and has no social concern whatsoever. This lack of social ability in NPCs can badly impact their believability and in turn affect the player's gaming experience.

While some work has been done to tackle the problem of believability in NPCs, survival games however, have not been subject of this effort.

1.2 Objectives

As our objective for this work, we propose the development of a platform that will enable the creation of NPCs controlled by agency based models for survival games. By making use of FAtiMA, an agency based model, and DST, a survival game, we will create a framework where agents can be created with FAtiMA and played in Don't Starve Together.

1.3 Contribution

With the conclusion of this project we've successfully developed the following assets:

FAtiMA-DST

A modification for DST that enables us to run FAtiMA controlled characters inside the game.

FAtiMA-Server

A console application the handles all the FAtiMA related processing for controlling characters.

Walter - The AI Companion

An example NPC published in Steam Workshop that makes use of FAtiMA-DST and FAtiMA-Server.

Tutorials & Instructions

A complete set of instructions publicly available in conjunction with FAtiMA-Server and FAtiMA-DST that enable developers to create their own FAtiMA powered NPCs for DST.

1.4 Outline

In Chapter 2 we will begin by exploring how the game industry is currently developing NPCs, then we'll explore some of the existing tools and frameworks for building agents, and terminate the chapter with the presentation of some agency models.

Then, in Chapter 3 we will focus on the game itself providing a comprehensive view of it's core mechanics, challenges, and describe a general play-through of the game. The conceptual model of the framework is presented in Chapter 4, while Chapter 5 depicts all the implementation details as well as a walk-through of the process of creating an NPC for DST, using the example character, Walter, as a base.

The results of this work are presented in Chapter 6. In this chapter we'll compare our example NPC, Walter, with an unpublished work from an anonymous *modder* and present a use case of our platform. Finally, in Chapter 7, the conclusions of this work are presented as well as some remarks on possible future work.

2

Related Work

Contents

During this chapter we will explore some concepts that are directly related to our work. Firstly, we talk about how the game industry is making NPCs, and what are the key aspects for the creation of believable characters. Then, we explore some of the existing frameworks and tools currently available for creating and testing agents for games, and multi-agents systems. Afterwards, we move onto presenting and analyzing some agent's models and architectures.

2.1 Games, Immersion and Believability

As we argued before, the increasing lifelikeness in video games gives players increasing expectations of better, meaningful experiences and interactions. The world around the player is presented with exquisite graphics and increasing levels of fidelity, as proved by the recent title from Guerrilla Games, Horizon Zero Dawn [11] (Figure 2.1).

However, NPCs still have a lot of room for improvement. Scripted behaviour and repetitive dialog, products of the lack of socially deep NPCs, can easily break the gaming experience of the player due to reduced believability in the NPCs actions [2]. This contrast between the world and it's inhabitants may break the suspension of disbelief, one of the major factors needed to achieve a successful gaming experience [3].

The concept of believable characters has long been studied and explored in many different forms of art and media [4]. From the original Disney animators of the 1930's and the book later published on Disney's animations [12], we can understand that in order to have believable characters we need to create the illusion of life.

2.1.1 NPCs in games

Most Triple-A video games make use of simple techniques to express NPC behaviour. Finite State Machines (FSM), hierarchical FSM and behaviors trees are the most commonly used techniques in nowadays industry [10]. However simple, they can achieve believable and expressive behaviours in the hands of skilled game designers. These techniques can express logic, basic planning and reactive behaviours.

Another advantage is that these techniques are deterministic, in other words, we can predict what can happen and then try to avoid problems. Their major weakness, however, is their limited expressiveness that can cause the realization of complex behaviors unmanageable.

In games like Assassin's Creed Unity [13] and Dragon Age Inquisition [14], where there is a large number of NPCs, there is a tendency to make them reactive to the players instead of actively following their own goals. In Dragon Age Inquisiton, many NPCs are present throughout the world doing absolutely nothing (check Figure 2.2). Sometimes, when the player approaches, she can overhear a conversation

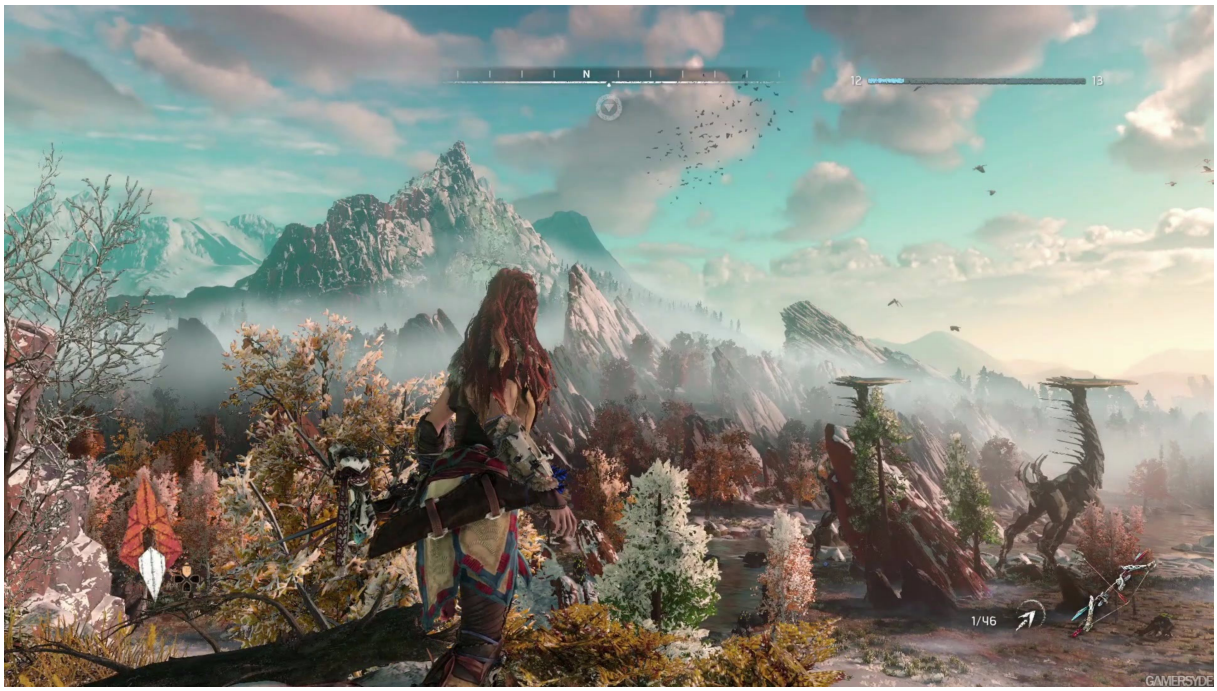

Figure 2.1: Horizon Zero Dawn exquisite graphics.

that is taking place between NPCs, usually directly related to the player, but these are always simple one or two sentences conversations and the player cannot interact with the NPC that is talking.

One can argue that, using current technology, it is impossible to make every single NPC socially aware and able to interact with the player. However, even if we only look at the NPCs with which we can interact, their behaviour is repetitive and lacks dept.

Other games, such as The Elder Scrolls V: Skyrim [15], have NPCs with occupations which they carry out, during the day (the blacksmith will spend her day at the forge working), and houses where they return to, during the night (almost every character will go to his home to have a meal and sleep). This simple behaviour highly contributes to the character's believability as it provides the sense of having wants and needs. However, they still lack social awareness.

If, for example, the player breaks into the house of a random NPC, wakes her up and has a talk with her, she will ignore the fact that you just woke her up in the middle of the night in her locked house and proceed to have a completely normal conversation with the player. The weirder part still, is that the player will be branded as a criminal and may face the city's guards upon exiting the house. This lack of social ability can easily break the player's suspension of disbelief.

To address this problem we can make use of research based agency models for NPC control, as has been done in other works [16] [9].

Figure 2.2: Example of Dragon Age Inquisition's NPCs.

2.1.2 Socially empowered games

There are, however, some examples of socially focused games that are worth to mention not only for their value as games but also for their successful use of research based agency models for NPC control.

Blood and Laurels

Blood and Laurels (Figure 2.3) is a text-based interactive drama available in the App Store for the iPad[1]. One of the game's appeal is its high re-playability due to the use of autonomous agents. The same episode can be played several times with different results. This is only made possible by making use of the Versu model [17], described in 2.3.2.

Blood and Laurels is set in ancient Rome and the player must weave through a web of conspiracies and politics. The game's success and critics are proof that the use of socially aware NPCs can produce highly adaptable and interesting games. The creators of Blood and Laurels are currently working on a second title called Bramble House which will also use Versu as the base for the autonomous characters (NPCs).

[1] You can visit the game webpage at https://versu.com/2014/05/28/blood-laurels/.

Figure 2.3: A screen shot of Blood and Laurels' game play.

Prom Week

Prom Week is a social simulation game about the interpersonal lives of a group of high school students in the week leading up to their prom [18]. Gameplay in Prom Week involves solving level goals (such as making the class nerd, Zack, date a popular girl) within a limited number of turns by directing the characters to engage in social interactions.

Which social games are available and how each changes the social state is managed by the game's AI system, *Comme il Faut* (CiF), described in 2.3.1. Enabled by the social physics of CiF, each level's goal has innumerable solutions that maintain character believability.

Mismanor

Mismanor is a social role-playing game where the player's main form of interaction are actions that change the underlying social model [19]. It makes use of an adaptation of the CiF architecture discussed in 2.3.1. In Mismanor, the player is guided through the dynamic story with quests that are chosen based on the actions the player has taken and their social standing with the character they are interacting with. Quests have multiple entry and exit points, giving the player more flexibility with how they choose to fulfill the request, with different consequences based on the path they have taken.

Façade

In Façade [20], the player is given almost no direction or role to play. The player can either play herself as the character or interpret the part of whomever she wishes. The drama takes place in a small simulated virtual world, the apartment of the married couple Grace and Trip. Façade was designed to deliver an experience that provides the player with 20 minutes of emotionally intense, undefined, dramatic action. The player's actions have a significant influence on how the story develops and how the drama ends.

Figure 2.4: The two NPCs of Façade greeting the player as she arrives at the dinner party.

2.2 Tools and Frameworks

During this section we'll talk about some of the frameworks and tools currently available for the creation of agents. We'll begin by presenting the well known Java framework Java Agent DEvelopment Framework (JADE) and the Foundation for Intelligent Physical Agents (FIPA) specification upon which JADE is based upon. Then, we'll talk of NetLogo and finally we'll present the PsychSim tool.

It is important to note that, to the best of our knowledge, there is, currently, no tool or framework that focuses in the development of NPCs for survival games.

2.2.1 JADE

JADE is a software framework in compliance with the FIPA specifications for interoperable intelligent multi-agent systems made in Java [21]. It aims to simplify the development of agents while maintaining standard compliance through a comprehensive set of system services and agents. JADE complies with FIPA's specification [22] which defines the rules that allow a society of agents to operate, inter-operate, and be managed. This specification is further detailed bellow, in 2.2.1.

While complying with the FIPA specification, JADE provides a set of abstractions that allow for the rapid development of agents, like the use of an abstraction model called Behaviour. Behaviours model the tasks that an agent is able to perform and each agent instantiates their behaviours according to their needs and desired capabilities. The framework also includes some ready to use behaviours for the most common tasks in agent programming, such as sending and receiving messages and structuring complex tasks as aggregations of simpler ones. The developer needs only to extend the Agent class

and implement the agent-specific tasks through one or more Behaviour classes, instantiate them, and add the desired behaviours to the agent.

FIPA

The FIPA specifications represent the first step towards an agent standard [22]. The specification does not attempt to define the internal architecture of agents nor how they should be implemented, but they do specify the interfaces necessary to support interoperability between agent systems. It identifies the roles of some key agents necessary for the management of the platform, and specifies the agent management content language, ontology, and agent communication.

Three key roles are identified as mandatory in an agent platform. The Agent Management System is the agent that exerts supervisory control over access to and use of the platform; it is responsible for authentication of resident agents and control of registrations. The Agent Communication Channel is the agent that provides the path for basic contact between agents inside and outside the platform; it is the default communication method which offers a reliable, orderly and accurate message routine service. The Directory Facilitator is the agent that provides a yellow page service to the agent platform. Notice that no restriction is given to the actual technology used for the platform implementation: e-mail based platform, Java applications, web services... these could all be FIPA compliant implementations.

Agent communication is based on message exchange, where agents communicate by formulating and sending individual messages to each other. The FIPA Agent Communication Language (ACL) specifies a standard message language by setting out the encoding, semantics and pragmatics of the messages. The standard does not set out a specific mechanism for the internal transportation of messages. Instead, since different agents might run on different platforms and use different networking technologies, FIPA specifies that the messages transported between platforms should be encoded in a textual form.

Other parts of the FIPA standard specifies other aspects, in particular the agent-software integration, agent mobility and security, ontology service, and the Human-Agent Communication.

2.2.2 NetLogo

NetLogo (Figure 2.5) is a multi-agent programming language and modeling environment for simulating natural and social phenomena [23]. Modelers can give instructions to hundreds or thousands of independent agents, called ”turtles”. These turtles can represent molecules, animals, people, bacteria, cars, robots, neutrons, magnets, planets, or whatever the modeler decides. The turtles move across a grid of ”patches” which are also programmable agents. All of the agents can interact with each other and perform multiple tasks concurrently. This makes it possible to explore connections between micro-level behaviors of individuals and macro-level patterns that emerge from their interactions.

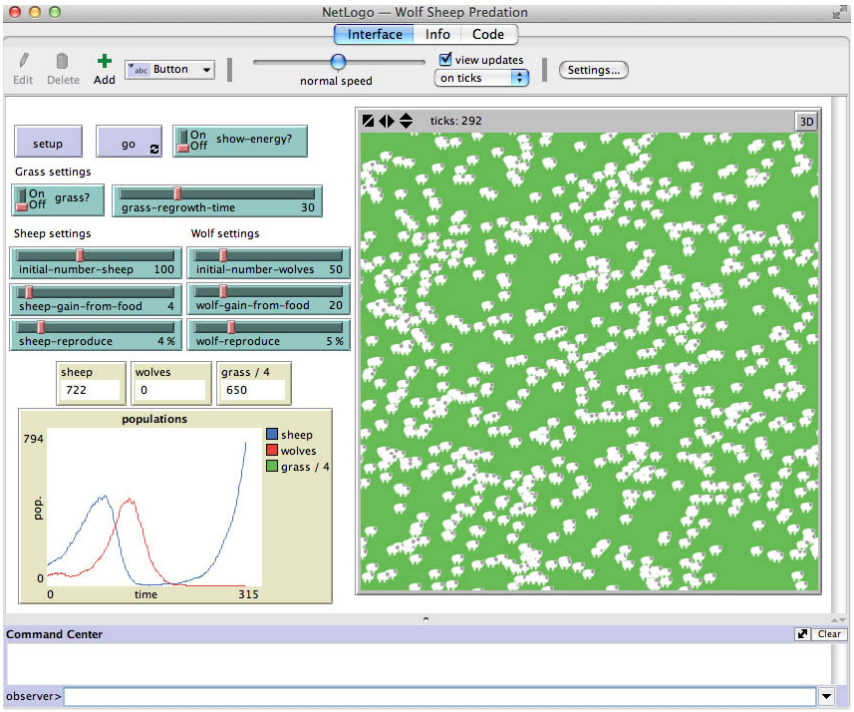

Figure 2.5: NetLogo's interface.

NetLogo comes with a Models Library, which contains several simulations. This collection has more than 140 pre-built simulations that can be explored and modified. These simulations address many areas in the natural and social sciences, including biology and medicine, physics and chemistry, mathematics and computer science, and economics and social psychology.

2.2.3 PsychSim

PsychSim [24] is a social simulation tool developed to explore how individuals and groups interact and how can these interactions be influenced. The work's foundation is the fact that social interactions are based on beliefs about other's minds, a *theory of mind* [25]. Psychsim allows developers to author and explore a social scenario of their creation through the selection of generic agents models and its further specialization, see Figure 2.6. This tool allows developers to try and understand how a specific social group might react to certain situations.

PsychSim then simulates the behaviour for each entity, individual agent or group of agents, based on their preferences, relationships, private beliefs, and mental models about each other.

In the given example, the authors explore a bullying scenario in a school and try to answer several questions: "*How might a bully respond to admonishments, appeals to kindness or punishment? How might other groups react in turn? What are the predictions or unintended side effects?*". The simulation tool then provides explanations of the results based on each entity's preferences and beliefs, allowing

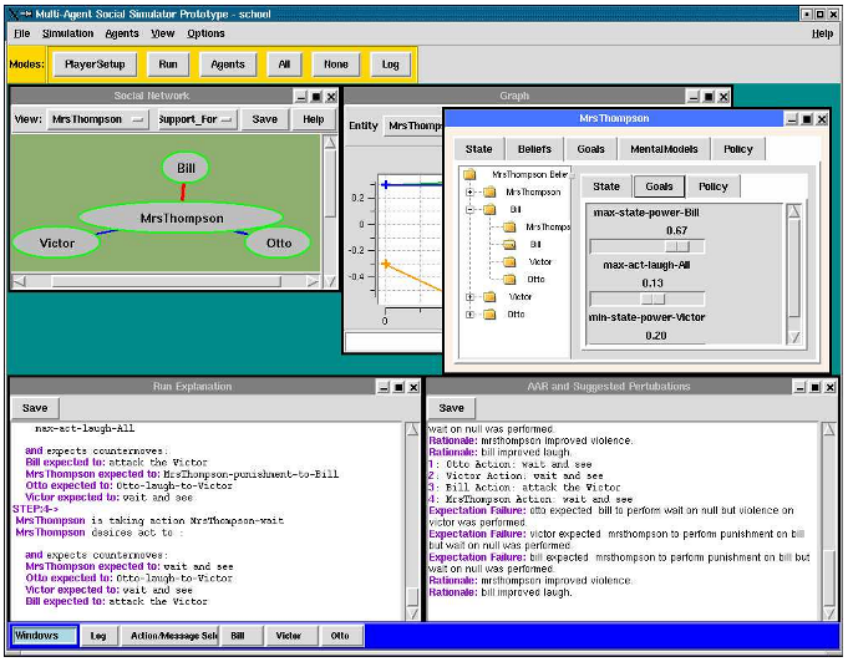

Figure 2.6: PsychSim tool for generating new scenarios and agents.

the authors to answer the previous questions.

PsychSim's agents are empowered with fully specified models of each other [26], a unique aspect of its design. Every agents maintains independent beliefs about the world, has its own goals, and it owns policies to achieve those goals. The model is composed by the state, actions, goals, beliefs, policies, messages, and mental models.

State Each agent model includes a state with facts about the world, some of which may be hidden from the agent.

Actions Agents have a set of actions they can perform. Each action consists in an action type, a performer, and possibly an object of the action.

Goals These represent an agent's incentives for behaviours. In PsychSim, goals are reward functions that map the current state to a real value.

Beliefs The simulations agent have only a *subjective* view of the world, where they form beliefs about what *they* think is the state of the world. An agent's beliefs consists in models of all agents (including himself), representing their state, beliefs, goals, and policy of behaviour.

Policies Each agent's policy is a function that represents the process by which it selects an action or message based on its beliefs and goals.

Messages Messages are attempts by one agent to influence the beliefs of recipients and have five components: a source, recipients, a message subject, content, and overhearers.

Mental Models An agent's beliefs about another agent are realized as a fully specified agent model of the other agent, including goals, beliefs, and policies. Mental Models are predefined models which represent an agent's goals, beliefs, and policies.

2.3 Social Architectures and Models

In this section we take a look into existing models for agency.

2.3.1 Comme il Faut and Ensemble

CiF is an AI system that enables authors to create interactive stories by specifying, not the complete narrative and all its ramifications but, high-level rules governing expected character behaviour given social situations [27]. In CiF, characters use many attributes of the current social state, including the story of prior interactions, to decide how to engage in social exchanges with other characters. This architecture provides a rich social environment for characters to interact allowing the creation of dynamic and interactive stories.

Social exchanges are the primary structure of representing knowledge in CiF [28]. They consist in social interactions between characters that modify the social state of the participants. By using social exchanges and additional encoded social context, CiF lowers the authoring burden needed to create the social aspects of an interactive story by allowing the author to specify the rules and general patterns of how social interaction should take place.

Characters' behaviour is chosen based on rules in a large rule database that depict normal social behaviour in a particular story world. These rules, in conjunction with the logic of a social world, a set of characters, and a series of scenario goals allows CiF to determine the desired action for each character.

More recently, a new version of CiF as been published and renamed to Ensemble. Ensemble aims to empower NPCs with deeper interactivity and believability by modeling social states and behaviours for game characters [29]. This extension presents a model of playable social dialogue called social practices. Social practices increase the playability of character interactions and add interactivity at each stage of dialogue.

2.3.2 Versu

Versu model is based on Exclusion Logic [30], a new deontic logic, and rests upon two kinds of objects: agents and social practices. To better make use of exclusion logic, the creators of Versu have also

developed Praxis, a Domain Specific Language (DSL) used in the modeling of the social practices .

Social practices describe a recurring social situation that may exist only for a short time (e.g. a conversation, a game, a meal) or can last much longer (e.g. a family, the moral community). These practices coordinate agents via the *roles* they are playing and their main function is to describe the actions the agents can do in each situation.

Social practices provide the agent with a set of suggested actions, but it is up to the agent to decide which action to perform, using utility-based reactive action selection (the utility is calculated in accordance with the agents beliefs, desires, personality quirks, and backstory).

Multiple practices can exist concurrently, for example, in a dinner party, there will be multiple practices operating at once:

- eating and drinking.
- the conversation about politics
- the rising flirtation between Frank and Lucy
- responding to the fact that Mr. Quinn has spilled the soup.

Performing an action can result in any sentence being added to the world database. The results of adding new sentences can be that relationships are updated, new beliefs or desires are formed, old practices are deleted or new practices are spawned. Agents and social practices are scripts authored using the DSL Praxis.

2.3.3 FAtiMA

FAtiMA is an agent architecture with planning capabilities designed to use emotions and personality to influence the agent's behaviour [31]. The acronym stands for Fearnot AffecTIve Mind Architecture as it was developed to endow the characters in the serious game *FearNot!* [32] with social intelligence. It has been used successfully in different social scenarios ([33], [34], [35], and [36]) and it is based on appraisal theory [2].

FAtiMA has a modular architecture where functionalities and processes are divided into modular independent assets. This enables developers to use a lighter and simpler version of FAtiMA by adding the required assets, according to their necessities. Therefore, behaviour and functionality are added by including assets in an agent's definition. These assets will then implement the desired behaviours and functionalities.

[2]The theory that emotions are elicited by evaluations (appraisals) of events and situations [37].

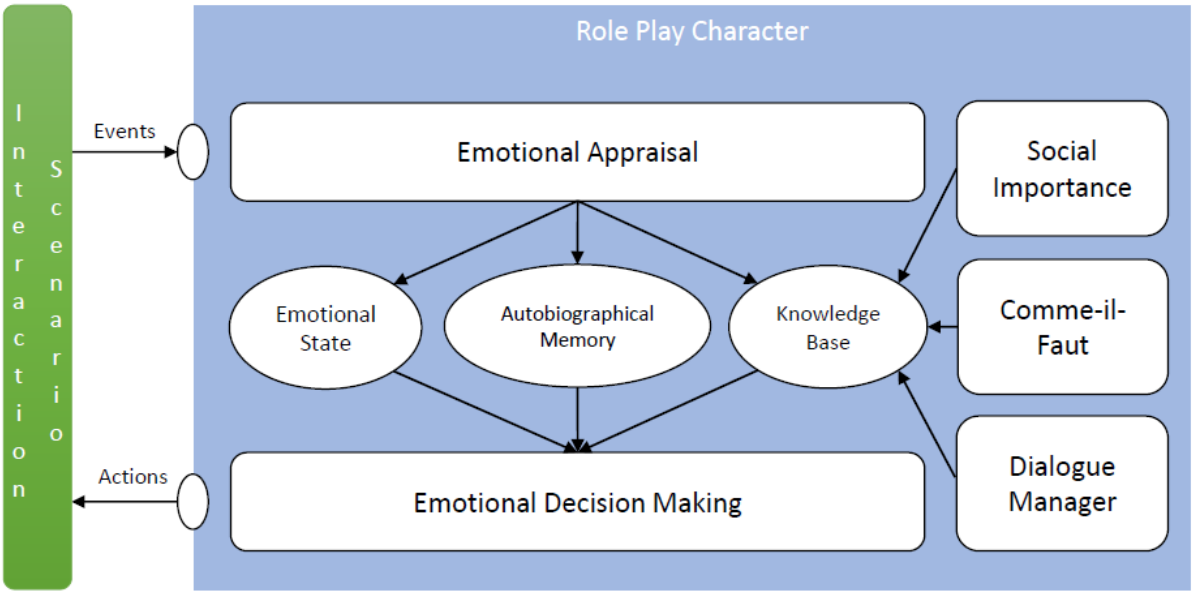

Figure 2.7: The Role Play Character Asset aggregating all other assets to make an autonomous agent.

FAtiMA comes with a set of predefined assets that can be used to define agents and their interactions[3]. Additionally, the developer can create its own assets that extend the toolkit's capabilities. The available assets are:

Role Play Character Asset

The Role Play Character (RPC) Asset is an aggregation asset responsible for managing all the other assets that compose an agent. It is also responsible for maintaining the agent's Emotional State and the agent's memory, which is divided in two components: a Knowledge Base and an Autobiographic Memory.

The Emotional State stores the current emotional state of the agent, defined by the Emotional Appraisal Asset, the Knowledge base stores the agent's beliefs, and the Autobiographic Memory stores the agent's recollections of past events and the emotions associated with such events.

Events are divided into three categories: Action-Start, Action-End, and Property-Change. The first two are self-explanatory and the third represents updates to the RPC's Knowledge Base. Therefore, in order to update the Knowledge Base, the RPC must perceive the appropriate events.

For consistency sake, both beliefs and events are described by Well Formed Names (WFN). These can be symbols (e.g. "`Walter`" or "`cutgrass`"), variables (e.g. "`[x]`" or "`[target]`"), and composed names (e.g. "`IsBusy(Walter)`" or "`Likes(Walter, [target])`").

Emotional Appraisal Asset

[3]The toolkit is available at https://github.com/GAIPS-INESC-ID/FAtiMA-Toolkit.

This asset appraises each event according to the OCC Theory of Emotions [38]. Developer defined appraisal rules, determine how an agent evaluates certain events, determining the agent's Emotional State (which is kept by the RPC).

For example, an agent, Mary, who likes John, might perceive the event of John flirting with her in a positive way, whilst another agent, Kate, who dislikes John, might perceive such flirtation negatively.

Emotional Decision Making Asset

The Emotional Decision Making Asset provides RPCs with decision making capabilities based on rules with logical conditions. Allowing the agent to make multiple decisions at the same time, this asset makes use of the Knowledge Base and the Emotional State to decide which actions it decides to do.

The possibility of making multiple decisions at the same time is particularly interesting when considering behaviour related decisions and non-behaviour related decisions (dialogue actions and/or emotional responses). FAtiMA Toolkit handles this by making use of what it calls layers of decisions. When the decision process is triggered, if a layer is provided, only actions for that layer are considered.

Additionally, this asset adds Dynamic Properties that can be used as conditions to actions. Dynamic Properties provide a facility for developers to implement additional processing capabilities for the decision of certain actions. For example, the developer could implement a *Gaze* Dynamic property that would connect to an external eye tracking device, in order to make the agent act in accordance to the user's gaze.

Integrated Authoring Tool Asset

This asset adds a Dialogue Manager that combines an hybrid solution between dialogue trees (a popular solution among game developers) and state machines to construct dialogues for agents. The Dialogue Manager keeps the current state of dialogue, giving an agent (or player) every available option to choose from. This is achieved through the implementation of the *ValidDialogue* Dynamic Property, that will advance through the Dialogue Manager.

Social Importance Asset

The Social Importance Asset implements the *SI* Dynamic Property, which represents the social relationship among two social entities A and B. This representation, based on the status-power theory [39], gives information about how willing an agent is of complying with another agent's wishes.

Note that the social importance relationship is not equivalent, meaning that the social importance that agent A gives to agent B might not the same as the social importance that agent B gives to agent A. The social importance values are maintained by a set of developer defined attribution rules that are evaluated after an event is perceived.

Comme il Faut Asset

Based in the work previously described in 2.3.1, this asset allows developers to define Social Exchanges which, unlike single dialogue acts, are composed by sequential steps: initiate, answer, and finalize. These steps represent an interaction between two agents.

Each of the steps share the activation conditions of the Social Exchange, and can have different styles (positive, neutral, and negative) that are calculated by the *Volition* Dynamic Property, introduced by this asset.

World Model Asset

The World Model Asset gives developers a way to test the created agent in a simulated world where the consequences of each action is defined by the developer. This asset is especially useful to test particular aspects of an agent.

By using the FAtiMA Authoring Tools, the developer can specify the complete current state of an agent and run it in this controlled environment. It also gives the developer a way to simulate the world the agent is in. which is particularly useful when we consider planning agents.

It is important to note that the CiF Asset can be used to complement the Social Importance Asset by giving an agent knowledge about social practices. For example, while a boyfriend and father might have equally high social importance values, flirting is not an appropriate social practice when the target is the agent's father. By defining flirting as a social exchange, the developer can define an additional condition to only activate this social exchange if the agent is attracted to the target.

2.4 Closing Remarks

The game industry is currently using outdated techniques to add behaviour to the characters that populate their games. Skilled game designers can still create interesting behaviours with the used techniques, but the need for social context and active goal pursuit requires more advanced techniques.

To build a platform that will enable developers to create collaborative and believable NPCs for survival games, we've, first of all, looked into frameworks and tools that focus on agent creation.

JADE provides a great platform for the development of agents. However, JADE's main field of applicability is that of software engineering. Its main purpose is the development of negotiating agents

that conform to the FIPA specification, having no concern whatsoever for interactions with humans, to such an extent that, although it is specified in FIPA, JADE is an agent-only platform with no support for Human-Agent Communication.

NetLogo is a great example of testability, which is one of our main focus. The ability to run fine tuned scenarios with different behaviours easily is a key point of its design and of utmost importance for our work, and that we will strongly value while developing our work. Despite that, it does not provide a model for agency which is one of the crucial points of this work.

The final tool we've looked into is PsychSim. As a social agent architecture, PsychSim presents a well-defined model capable of fully simulating social behaviour, is based on sound theories from psychology, and takes into account other agents in its cognitive process. But, it lacks the possibility to have agents interacting with humans. Additionally, as the agent's behaviour is based only in policies which guide it's immediate behaviour, each agent will follow the scenario devised by the developer without deviation.

We've then moved into exploring some of the currently used agency models. The three systems we described were CiF, Versu, and FAtiMA.

Despite the successful implementation of CiF in the game Prom Week [18], the game developers still need to specify every rule for every possible social interaction between NPCs. Prom Week contains over forty nine hundred unique influence rules, around sixty social exchanges with over twenty rules that contribute to the characters' desires and responses, a cast of eighteen characters, and a combined total of over forty thousand predicates. This represents an authoring burden that we want to avoid when developing our NPCs. Even its successor, Ensemble, which lowers the authoring burden by adding social practices, has been recognized, by the authors, to possess limitations: a social practice must be taken to completion before another practice can be initiated, which does not reflect real social interaction.

Versu's logic powered approach is a differentiating aspect from the other models, however it does imposes some limitations. One example is how the agents beliefs are expressed. The system cannot represent universal and existential quantifiers (e.g. "everyone has become insane" and "the murderer is one of the guests", respectively) or beliefs about others' beliefs (e.g. "Mr Quinn believes that Lucy believes that Mrs Quinn is the murderer").

Finally, and although FAtiMA, by itself, is not a determined model for agency, it provides a set of ready to use assets, that allow developers to use whatever they need to achieve their goals. The inclusion of emotional appraisal, theory of the mind models, and its planning capabilities make FAtiMA the best candidate to be used. This, allied with the fact that it has been successfully used in several social scenarios with human interaction ([33] [34] [35] [36]), the ability to extended the existing assets, and the ability to incorporate planning into its decision making process, make it the best choice for our work.

3

Case Study: Don’t Starve Together

Contents

Figure 3.1: Don't Starve Together poster with all original characters.

Don't Starve Together is a multiplayer wilderness survival game developed at Klei Entertainment, where the players must survive as long as they can. The players must face the harshnesses of a procedurally generated world that is actively trying to kill them, either by cooperating or competing with each other. Each game can last and indefinite amount of time which is only determined by the players' ability to survive: the better you play the game, the longer you can last.

Released on April 21st of 2016, DST is the standalone multiplayer expansion of the uncompromising wilderness survival game Don't Starve, also released by Klei on April 23rd of 2013 (check Fig. 3.1). Both titles are available in several platforms ranging from PC's to consoles and even mobile devices. By the end of 2013 the original title (Don't Starve) sold over one million copies and, currently, there are almost 3.5 million owners of the game only on Steam. DST counts with over 7 million owners on Steam and has a daily peak of concurrent players of about 8 thousand, reaching as high as 12 thousand concurrent players.

The game counts with a developer maintained forum where there is an active participation by Klei Entertainment staff, which help in problem solving for community developed mods, bug fixing the game, and sharing development notes on the game. Currently, over five thousand mods are available for the game on the Steam Workshop.

As we've previously discussed, survival games offer players different challenges that must be overcome by the character in order to survive. Some do it by providing mechanics for hunger and thirst while others provide horrific scenarios like zombie infested worlds.

DST borrows ideas from both sides. The game presents mechanics for hunger, health, temperature, wetness, day and night cycles, seasons, and sanity. And it also provides players with a monster infested world that try to kill the player's character. The addition of multiplayer brings the possibility for social interaction among players, which contributes positively for the gameplay experience.

Moreover, the sanity mechanic itself is a differentiating factor from other games in the genre. By providing a measurement of the characters' mental health, the game makes the characters more human and relatable. A normal human being under such harsh environments would suffer physically but also mentally, and the sanity mechanic depicts just that.

For these reasons, the lack of AI controlled NPCs with believable behaviour, the ability to control and personalize the generated game world, and the possibility of introducing modifications into the game, we've decided to use this game as the target of our project.

For the remainder of this chapter we will present a thorough break down of the game. We'll explain what the game consists in and move on to describing the game's world. Then, a general gameplay experience is briefly described before ending the chapter with a summarization of the game mechanics.

3.1 The Game

Before playing DST, the player must first generate a game world to play in. The generation process will create a new world in accordance to a set of configurations. The configurations allows her to personalize every aspect of the game world, including the world size, the abundance of resources, the existence and frequency of monster appearances, and many more options.

The player will then choose a character from a list of available characters. Different characters have different characteristics and may be affected by the world in different ways (for example, WX-78 can eat spoiled food without penalties) and can even possess special items (such as Woodie's axe, *Lucy the Axe*). Upon entering the world, the player has nothing in its inventory (except for character specific items) and thus needs to start collecting resources in order to survive.

During the game, the player must pay attention to the three characteristics of her character: Hunger, Sanity, and Health. The player dies (in DST the player actually becomes a ghost that can be revived by other players or by finding a Touch Stone) once his Health reaches zero. Both Hunger and Sanity can cause the player to loose Health (Sanity doesn't directly cause the player to loose Health, but the player will be attacked by creatures that appear due to the character's paranoia).

So, it is important for the player to keep Hunger, Sanity, and Health in check in order to survive. The player will also need to take attention to the character's Wetness which is caused by rains. Wetness will cause the items to loose efficiency and by using wet items the character will lose Sanity over time. The player must also be aware of the temperature of his character as it can cause damage. Both extreme

Figure 3.2: Example of a generated map of DST.

hot and extreme cold will cause the player to continuously loose Health.

Taking care of the character's Hunger, Sanity, and Health is the player's main concern during gameplay. She will accomplish this by collecting resources, crafting new items, cooking food, among other things (we will explore the player's possible actions in full detail further on).

3.2 The World

As mentioned earlier, the world is procedurally generated for each game and has several different biomes that provide different kinds of resources but also dangers for the player. We'll present and describe these biomes later on.

The world is surrounded by water and is, in many cases, composed by several peninsulas which can be composed by one or several biomes, as can be seen in Fig. 3.2. It's also common to see the formation of lakes inside the main portion of land.

During gameplay, time will pass not only in a night and day cycle but also across seasons. The game has four seasons (Autumn, Winter, Summer, and Spring) that present different challenges during the course of the game. In Winter, the temperature descends a lot, while in Summer, the temperature rises, causing the player to take damage over time due to extreme temperatures. In Spring, the commonly occurring rains will increase the character's Wetness, which will have the effects already described. Autumn is by far the friendlier season, with mild temperatures and occasional rains, making it the perfect

Figure 3.3: Deerclops, the Winter giant, chasing after Webber.

candidate for the starting season.

Seasons bring yet another challenge for players: giants (see Fig. 3.3). During each season, a giant (for each season there is the corresponding giant) will spawn in the vicinity of the player and will attack him. The player can either choose to engage in a fight with the giant or try and lure him away until the giant looses interest.

A normal world will also have caves that the players can descend to, but we will not discuss them here since we won't consider them for our work.

Moving on to the specifics of each biome, we'll now explore a list of all the available biomes on DST and a brief description for each of them:

- **Chess** this biome has an abundance of marble (a very rare material in the game), however it is protected by aggressive mobs;
- **Deciduous Forest** being a forest, this biome is a great source of wood but also fireflies and mushrooms;
- **Desert** can be a good source of grass and twigs (two of the most important resources on the game) but can also be very dangerous due to the Hound mob spawn points;
- **Forest** a very good source of wood but can also contain spiders;
- **Grassland** one of the safest biomes where the player can find a good variety of resources;

- **Graveyard** can contain many gold nuggets and can be hazardous free. By digging up graves the player can collect loot but there is a chance that a Ghost may spawn and attack the player.
- **Marsh** commonly known as Swamp, is the harshest biome in the game, everything here will try and kill the player;
- **Mosaic** this biome typically appears only once per map, and is a mix of all other biomes with not so great resources;
- **Savanna** the abundance of grass and mobs (passive mobs like Rabbits, Birds, and Beefaloes) make this biome the best for the player to settle. Although it can also become a dangerous place during the Beefaloes mating season where they become aggressive;
- **Rockyland** a barren biome that contains a lot of boulders but almost nothing else. It's a common place for Tallbirds, an aggressive mob.

The knowledge of each of these biomes will differentiate good players from expert players, as it can strongly influence their success. Players must know where to for look what they need, where to settle, places to avoid, among others.

For the remainder of this section, we'll explore the environment of DST which we just described in a more formal way.

According to Russel and Norvig's classification of environments [40], DST is a partially observable, multiagent, stochastic, episodic, dynamic, continuous, and known environment.

Partially observable In DST the environment is actually fully observable through the scripts it natively provides to the modders and every entity is directly accessible to the agent. Despite this, we've decided to limit the observability of the environment to keep the agent in the same level of knowledge with the player, therefore, it will only have access to the same area as seen by the player in the screen.

Multiagent The world may contain several players, and is therefore a multiagent environment.

Stochastic During the game, spontaneous rains can occur, meteors can fall from the skies, and even lightning strikes can set the players crops ablaze. This all happens randomly and the agent will need to act accordingly in order to survive.

Episodic Although the world is in a continuous flow, we can consider it an episodic environment. Due to the continuous passage of time, day after day and season after season, the player will have to overcome recurring problems. Every night the player will have to deal with darkness (which can kill the player) and every season similar challenges (Winter always presents the same challenges).

Dynamic During the deliberation process of the agent, the environment around him can change without direct interference from it, be it through the passing of time and therefore seasons, or be it by the aggressive and passive mobs that populate the world (e.g. Moleworms search for and eat flints, one of the most valuable resources in the game)

Continuous In DST time passes continuously without interruptions and even actions take time to achieve, e.g. when collecting resources there is an associated animation that must be completed for the agent to effectively receive the resources.

Known The laws that rule DST are all known to the agent and therefore the outcome for every action is known.

As described above, the world in DST isn't a trivial one and will therefore represent a strong challenge when developing an NPC. However, the rules that apply to each generated world may vary and can be customized. Before starting the game, the player can tweak with a set of different options to personalize the resulting world.

Tweaks vary from very simple stuff, like increasing the rate at which hounds attack the player, to game changing stuff, like disabling all monsters in the game. This ability to personalize the world, is crucial to creating worlds with various degrees of challenge for the players and possible NPCs.

3.3 Gameplay

In order to survive, players must keep Health, Sanity and Hunger in check, as described before. To achieve this, players must collect resources, craft weapons, tools, and equipments, fight, and find a home. For the remainder of this section we'll explore, in a general view, what does a player do when playing DST. This will allow us to identify what sort of actions our agent will perform and group them into specific problems. Before the end of the section, we'll have a comprehensive view of the problem which will allow us to better understand how can we solve it.

The first few days in DST are the most crucial ones. Players will usually collect flint and twigs, used in the construction of tools like axes and pickaxes, in order to collect logs (to build a fire) and gold nuggets (very important for prototyping new tools and weapons). Finding food is of the utmost importance. Players usually rely on berries and carrots, which can be easily caught during the first days. Collecting cut grass is also of the most importance given that the players will need to build traps in order to keep stockpiling food.

Along with all the collecting, players must explore the world, with the intent of finding a resource filled area where they will later set up a base camp. Usually, by the end of the fifth day, players will have found a suitable place to build a base camp and will have found at least a gold nugget.

Figure 3.4: Wilson fishing in a pond.

In the beginning, players will only have the knowledge to craft basic tools and equipment, e.g. axes, pickaxes, hammers, Grass Suits (armour), Straw Hats, among others, in a total of fifteen recipes. By building a Science Machine, which will consume a gold nugget, players will be able to prototype new items, thus learning the recipes for them. After prototyping a new item, players are able to freely craft that item, given that they have the required materials. As the game progresses, players learn up to a total of over one hundred and fifty recipes.

As for the base camp, players tend to find places which can provide great quantities of meat since it will be the base of the players diet after the first few days. Rabbit holes and Ponds are usually good places, as every day they spawn Rabbit and Frogs, respectively. Additionally, ponds can be harvested with a Fishing Rod in order to collect fishes (Fig. 3.4), that once cooked in a Crock Pot will provide a good Health regenerative. Another important factor in the choosing of a base is the proximity to manure, a must have resource for the building of farms.

As the time passes and players start to have a steady supply of food items, the tendency is to start relocating other resources to make them more accessible. Most players will dig up Grass Tufts, Sapplings, and Berry Bushes (they provide Grass, Twigs, and Berries, respectively) and plant them near the base camp, as seen in Fig. 3.5. This will greatly increase a players collecting rate as all will be

Figure 3.5: An example of a base camp.

concentrated in an area around the players' camp, whereas the natural occurrence of such resources is usually sparse (occasionally, Berry Bushes and Grass Tufts will spawn in small clusters of nine elements near Pig Villages). However, the relocation of Grass Tufts and Berry Bushes is usually troublesome as they require to be fertilized (using manure) in order to regrow.

In order to feed themselves, players use the Crock Pot to cook the food. Crock Pots allow the elaboration of dishes that have several beneficial characteristics. They can restore Hunger, Health and Sanity, with different dishes having different values, e.g. Fish Sticks restore a lot of Health but don't restores as much Hunger while the Meaty stew does the opposite. For the cooking of dishes players use meats and vegetables. The later is often obtain from farms, hence their presence in a players' base camp.

The base camp will usually be composed by a Science Machine or an Alchemy Engine (an improved version of the Science Machine), Farms, Drying Racks (used to dry meat which can them be stored for a longer period), Crock Pots, Chests (used to store items), clusters of Tress, Grass Tufts, and Saplings, and one or more Firepits.

As said before, if players stand in complete darkness for too much time, Charlie, the antagonist of the game, will attack and kill the player. By standing next to a source of light, like a campfire, the player avoids being attacked by Charlie.

3.4 Closing Remarks

We've now thoroughly explored DST game mechanics and frequently faced adversities. A player's main concern is to keep the character's stats balanced throughout the game and find ways to survive the dangers each season brings. To keep a steady supply of food and protection from the elements is as important as defending themselves from the monsters that will try to kill them.

To wrap up this chapter, bellow is a list of the core game mechanics with a small description. This list represents a compilation of key aspects we have to take into account while making an NPC for DST.

Health

When it reaches zero a character dies. It can be replenished by certain items and by eating appropriate food and will be lost while fighting and when in extreme conditions of temperature.

Hunger

Will cause a character to start losing health when it reaches zero. It will gradually decrease over time and can be increased by eating food.

Sanity

Represents the mental state of the character. Certain conditions will cause it to decrease (e.g. fighting or standing in the dark) while others can cause it to increase (e.g. eating certain foods or being around befriended *pigmen*). When near zero will provoke hallucinations that can attack the character.

Temperature

In cases of extreme cold or heat the character will start to lose health over time. It can be kept in check with the use of craftable items like clothes and fires.

Wetness

As the character gets herself wet it can drop held items, food in the inventory will spoil faster, and her temperature will also decrease. The use of umbrellas and appropriate clothing will prevent the wetness level from rising.

Darkness

When in complete darkness a character will be attacked by the darkness creature. It is important to stay near a source of light during the night.

Fighting

Either as a means of defense or of gathering food, fighting is a core game mechanic without which survival is impossible.

4

Conceptual Model

In this chapter we present a conceptual representation of our work. Throughout this chapter we will describe each of the components of the model and its expected behaviour. By the end of this chapter, the transition from the model, presented here, to the implementation, presented in Chapter 5, should be clear.

In Figure 4.1 we can see several different modules and concepts. The presented decoupling between the agent's AI and the agent's embodiment, allows us to consider the use of different AI techniques and different survival games.

Do notice that, despite the fact that the agent's AI is represented outside the visual representation of the survival game, it needs not be the case. Both concepts can be as tightly or as loosely coupled as needed for each different implementation.

Nonetheless, the Survival Game will need to provide ways for an NPC to interact with the world, both by collecting information of the world using sensors and by making changes to the world through the use of actuators. The NPC will then extract the necessary perceptions about the world and, via a specified channel of communication, send them to the appropriate AI. The AI will then perceive the received perceptions. Whenever necessary, the AI must emit actions that will be executed by the NPC via the use of its actuators. Throughout this execution, a Logger component will register all the executed actions, their outcome, and all the perceptions sent to the AI.

Now that we've explored the general flow of the model, we'll describe each of its components in detail.

AI Agents

This represents the AI that will effectively control an NPC. It is responsible for receiving perceptions and make decisions. It must be able to handle behaviour decisions and non-behaviour related decisions (such as dialogue).

This module will need to have reactive capabilities as well as strategic decision making. Due to their dangerous environments and sometimes monsters, survival games (and in specific Don't Starve Together) require the ability to handle unexpected events, e.g. when a monster comes to attack the character. Additionally, they also require strategic decision making in order to better survive the world. The harshness of seasons will require that characters prepare ahead to better survive.

Additionally, the capability for dialogue should be considered. Great part of player to NPC interaction is based on dialogue, as such, this module should be able to handle dialogues.

Dispatcher

This module represents a bridge between the AI and the embodiments of agents, managing the creation and destruction of the necessary AI modules in accordance with the existing embodiments

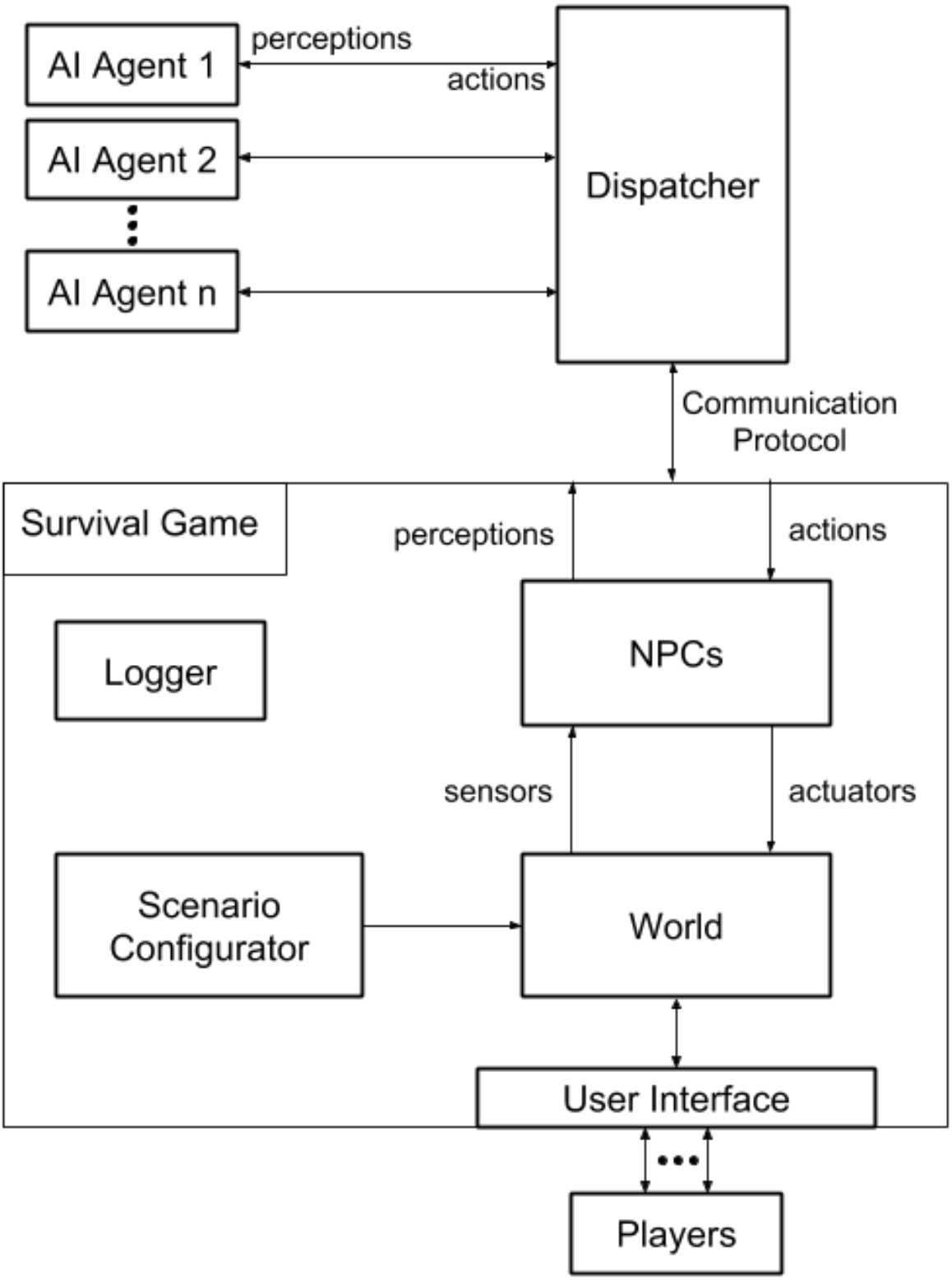

Figure 4.1: The proposed conceptual model.

(NPCs). It is responsible to direct each perception and action to the appropriate recipient, be it the AI or the embodiment of the agent.

Communication Protocol

Only present for illustrative purposes, the communication protocol is the bridge where perceptions and actions travel back and fort. This can be a simple local method invocation or a more complex communication system such as a remote method invocation over the network.

Preferably the former, which presents significant advantages: reduced response time, no worries about availability of external services, and no need to translate information between potentially different systems. But, in the case of the later, it must handle the translation of information between systems transparently.

World

The virtual world of the game. This is where all the characters (NPCs and player characters) actually live and must provide ways for the NPCs to act upon it and perceive it.

Scenario Configurator

This is, ideally, a part of the survival game. It provides a simple set of configurations that a developer can specify to configure the scenario that will be executed. It can also incorporate some configurations for the AI that will empower the NPCs.

NPCs

This module represents the embodiment of the agents. It will execute any actions on the world and collect the necessary perceptions which in turn are sent to the AI.

Logger

The Logger will be used to keep a registry of all the executed actions and their outcomes. It will also keep the registered perceptions of the NPC. Although its representation puts it as a part of the game, this module can be independent from the game itself.

User Interface and Players These modules are illustrative of the presence of players characters in the world. The survival game must accommodate the presence of several players that can play cooperatively or competitively and communicate with each other and the NPCs.

5

Implementation

Contents

To demonstrate our model, we've decided to implement it using a published commercial game, Don't Starve Together and a socially empowered model for agency with proofs given, FAtiMA Toolkit. During this chapter we will explain the inner workings of both Don't Starve Together and FAtiMA, and then describe how we managed to implement our model.

In the end of this chapter we will also describe an NPC that we published for the Don't Starve Together community, that can be played with. After reading this chapter, the reader will have the necessary understanding to use our framework to develop an NPC for Don't Starve Together.

5.1 Don't Starve Together - The Survival Game

As described in 3, Don't Starve Together is a multi-player survival game, in which players either compete or cooperate in order to outlive the opponents and the world. After reading that section, you will have an understanding of the challenges players face when playing the game.

However, we need to also understand the inner workings of the game. For the remainder of this section we will explore the game's internal organization in detail, exploring how we can introduce our modifications into the game.

5.1.1 Game Modifications

In Don't Starve Together, modifications are introduced in the game through the use of the Steam Workshop, a public platform, where players can share modifications for games. These modifications alter the original content of the game, adding new characters, creating new challenges, but mostly, altering the way the original game works.

In this particular case, the *modders* (the name given to the players who share content on the platform) can alter the content of the game by writing Lua scripts[1], which are loaded by the game and modify its content (https://www.lua.org/about.html). The game itself is already prepared to receive these *mods* (the name given to the modifications published to the platform), and while some kinds of *mods* are somewhat discouraged (like the addition of skins for the games characters), others are encouraged by the developers, who actively support the community in the game's forum.

For a *mod* to be loaded by the game, it has to follow a specific structure and use of a set of helper functions to include the alterations. As the game content is all written in Lua scripts (which are available to whomever owns a copy of the game)[2], *modders* are encouraged to check the original game files in order to understand how the game works and how to implement their own changes.

[1] Lua is a scripting language and is currently the leading scripting language in games
[2] The game engine is written in C++, but all the content and functionality is loaded through these Lua scripts

5.1.2 Entities

Everything that exists in the world of Don't Starve Together is represented as an Entity, from the character the player controls, to the sounds the player hears. These, in turn, are instances of *prefabs* that can be accessed and edited in the Lua scripts used to make the *mods*. It is important to note that each entity is uniquely identified by a six digit Globally Unique Identifier (GUID).

By specifying and configuring its *components*, *stategraphs*, and *brains*, the *prefabs* define every entity in the game. Each of these parts represents something different: *components* represent what an entity does; *stategraphs* represent how it looks; and *brains* represent how it behaves.

When a *prefab* is defined, it also configures its *components*, for example, in the definition of the *prefab* "wood", which contains the *component* "fuel", there is also the configuration of this *component* that, in comparison to the *prefab* "charcoal", has a lower level of "fuel". Effectively, *components* describe functionalities that a given entity can have, and how that functionality works. When an entity, either by a player's command or the *brain*'s command, does a chop action, it is the *components* of the held axe that determine how this action will be executed and trigger the appropriate animations and events.

While *components* can be reused throughout different *prefabs* (with different configurations), *brains* are defined on a per *prefab* basis. Both butterflies and spiders share the "locomotor" *component*, but have distinct moving behaviours.

When defining the *prefab*, if it is meant to have behaviour, an instance of a *brain* is also defined and attached to the *prefab*. However, not all *prefabs* have autonomous behaviour, and those that don't, don't have a *brain*. The *brain* itself is a Behaviour Tree that is attached to an entity, telling it what to do, leaving the part of how to do it up to the entity's *components*.

Using the Lua scripts, a *modder* can create *components*, *stategraphs*, and *brains*. These can then be attached to existing *prefabs* or new ones, also created by the *modders*.

5.2 FAtiMA Toolkit - The AI Agent

For the implementation of the AI we chose to use FAtiMA Toolkit [31], an agent creation toolkit. As previously discussed in 2.3.3, FAtiMA has been successfully used in several social scenarios ([33] [34] [35] [36]), making it the best choice for our implementation.

In this section, we present the Authoring Tools available to work with the toolkit. Throughout this section, we will use the term developer to refer to someone who uses the FAtiMA-Toolkit to create agents.

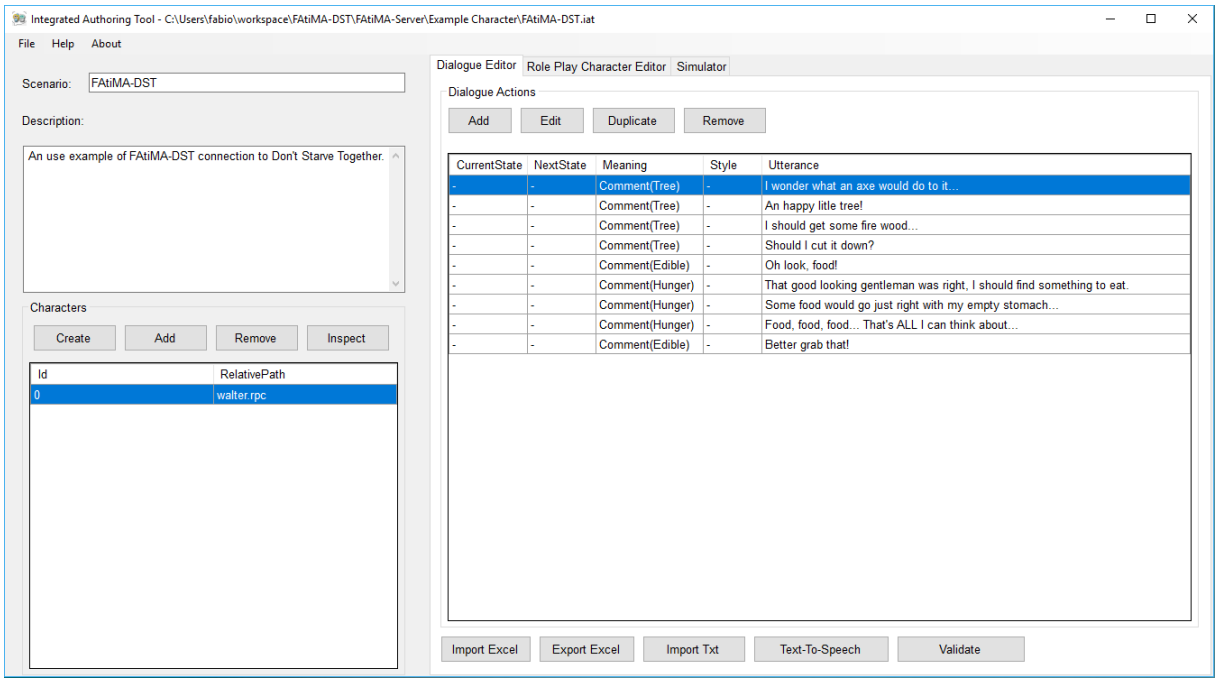

Figure 5.1: The Integrated Authoring Tool for FAtiMA Toolkit.

5.2.1 The Authoring Tools

The easiest way to develop a character with FAtiMA Toolkit is by using the provided Authoring Tools. Every implemented asset (described in 2.3.3) comes with a Graphical User Interface (GUI) Tool which provides facilities to help develop a character.

The main tool is the Integrated Authoring Tool (IAT), which can be seen in Figure 5.1. Through the IAT, developers can access every other asset to develop their agents. Developers can create dialogues, RPCs (and its components: Emotional Decision Making, Emotional Appraisal, etc.), run a chat simulator, and simulate and inspect RPCs.

In Figure 5.1 a set of dialogues from the example provided with our platform, Walter, can be seen. The developer can use the example provided to create her own RPCs, and then use the Authoring Tools to test them. The RPC Inspector gives developers the possibility to provide a set of events that will be perceived by the RPC. The developer can then execute the contained scenario to see the emotional appraisal and decision processes in action.

5.3 Putting it together

Now that we have explored both the game used as a scenario and the technology used to control the characters, its time to look at how we connected both. For the rest of this section, we'll present the

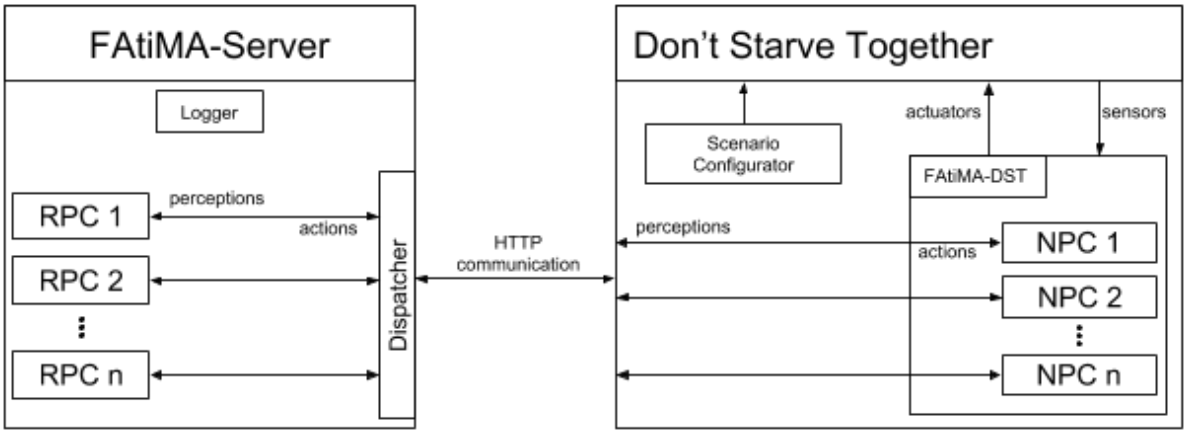

Figure 5.2: A graphical representation of the implementation.

implementation of the framework, explaining in full detail the decisions made.

The implementation itself consists in two modules: FAtiMA-DST, a Don't Starve Together *mod*; and FAtiMA-Server, a C# console application. Both these modules have been made publicly available on the Github repository https://github.com/hineios/FAtiMA-DST. In the repository there is also a guide to help anyone develop an NPC for Don't Starve Together. As shown in Figure 5.2, the communication between the two modules is made using Hypertext Transfer Protocol (HTTP), which transfers JavaScript Object Notation (JSON) objects back and forth.

The choice of using HTTP communication arouse from a limitation we found while developing the initial proof of concept for this implementation. Initially, we considered importing the FAtiMA Toolkit directly, as the loading of C# Dynamic-Link Library (DLL) files into a Lua interpreter is possible. However, as a security measure, the Lua interpreter embedded in DST, blocks any importation of external libraries. The inclusion of malicious libraries through the Steam Workshop would be a major security issue for both the company and the players.

As alternatives, we considered using textual files or a relational database that would act as a proxy between the modules, but we finally decided to use an HTTP connection as the game provided an easy way to make requests and register callback functions to handle the responses. Due to the client-server nature of HTTP however, the FAtiMA-DST module continuously makes HTTP requests with JSON encoded data to the FAtiMA-Server module, which handles these requests yielding an appropriate response (also JSON encoded). As a data-interchange format, JSON provides the necessary abstraction to exchange information between C# objects and Lua tables.

5.3.1 FAtiMA-Server

The FAtiMA Toolkit is written in C# and published as a set of DLL libraries. Taking this into account, we decided to implement a simple server in C#, capable of handling the HTTP requests made by FAtiMA-DST *mod*. This server, contains a list of Role Play Character Assets that are uniquely linked with the agents running on the scenario. Each request that the server handles, has an entity's (NPC) GUID, which is used to also identify a Role Play Character Asset.

Additionally, requests are divided into three categories: perceptions, events, and decisions. Perception requests contain information on the NPC's state and field of vision, and happen periodically. Event requests contain information on relevant world events, and happen whenever an event is triggered. Decision requests trigger the decision process that tells the NPC what to do, and happen periodically.

If we consider the flow of information, the first two types (perception and event requests) represents a flow of information from the embodiment to the AI, while the third (decision requests) represents a flow of information from the AI to the embodiment.

When received, perception requests are translated into a series of Property-Changed events, that update the RPC's Knowledge Base. Event requests will be translated to Action-End events and decision requests will trigger the RPC's decision process for the appropriate layer.

It is important to notice the limitations of using HTTP as the method of communication. This form of communication makes it impossible for the AI to give direct actions to the embodiment, as we cannot send messages from the server to the client, only respond to client's requests. As such, we have to rely on periodic requests made by the embodiment that will cause the AI to make a decision and respond with an action.

5.3.2 FAtiMA-DST

This module was implemented as a *brain* for DST and is responsible to send perceptions to FAtiMA-Server and execute the actions it returns. To achieve this, it sends a perception request to FAtiMA-Server two times a second, a decision request for behaviour every one and a half second, and a decision request for dialogue every ten seconds. The values were tested and tweaked for the best compromise between behaviour and performance.

To send the perceptions to the FAtiMA-Server, it counts with a set of helper functions that extract the necessary information from surrounding entities. A set of listeners relay information about events to the FAtiMA-Server, and a Behaviour Tree executes actions received from the FAtiMA-Server. For a complete list of beliefs check Tables 5.1 and 5.2 and for actions refer to the Appendix A

The Behaviour Tree makes use of the built-in Buffered Actions to execute the actions. These provide a layer of abstraction over actions to handle all the locomotion details and appropriate verifications (e.g.

it is only possible to chop a tree if an axe is held).

To handle the FAtiMA-Server commands, the tree contains two nodes: a node that executes actions (via buffered actions), and a node that handles the special behaviour of wandering. The handling of dialogue is independent from the behaviour and is executed immediately without making use of the behaviour tree.

5.4 Creating Walter

As an example NPC, we've implemented a model based agent to demonstrate how one could use the set of available beliefs to create behaviour. This example only makes use of FAtiMA's Dialogue Manager and Emotional Decision Making assets.

This example has been published to the Steam Workshop in the form of an AI companion, an NPC named Walter [3]. On the *mod*'s public page, a set of instructions can be found on how to run the character. The page also contains links to the public repository where all the code for this framework can be found.

The current set of beliefs available can be divided into two groups: character's state beliefs and world's state beliefs. The complete listing of the beliefs is presented in tables 5.1 and 5.2. Most of them are self explanatory, but others require some explanation.

Table 5.1: Character's state beliefs

Belief	Description
`Health([name]) = [value]`	Describes the agent's (*name*) health
`Hunger([name]) = [value]`	Describes the agent's (*name*) hunger
`Sanity([name]) = [value]`	Describes the agent's (*name*) sanity
`Moisture([name]) = [value]`	Describes the agent's (*name*) moisture level
`Temperature([name]) = [value]`	Describes the agent's (*name*) temperature
`IsFreezing([name]) = [bool]`	Describes if the agent (*name*) is taking damage from extreme cold
`IsOverheating([name]) = [bool]`	Describes if the agent (*name*) is taking damage from extreme hot
`IsBusy([name]) = [bool]`	Describes if the agent (*name*) is currently executing any action
`PosX([name]) = [value]`	The agent's (*name*) current X position
`PosZ([name]) = [value]`	The agent's (*name*) current Z position
`InLight([name]) = [bool]`	Defines if the agent (*name*) is in the light or darkness
`InSight([GUID]) = [bool]`	What the agent (*name*) is currently seeing
`InInventory([GUID]) = [bool]`	What the agent (*name*) has in his inventory
`IsEquipped([GUID]) = [bool]`	What the agent (*name*) has equipped

The `IsBusy` belief is used to represent if the NPC is executing an action. As the implementation stands, if an action is received by FAtiMA-DST, the NPC will immediately execute that action. This

[3] The *mod* public page can be found in this link: http://steamcommunity.com/sharedfiles/filedetails/?id=1339264854

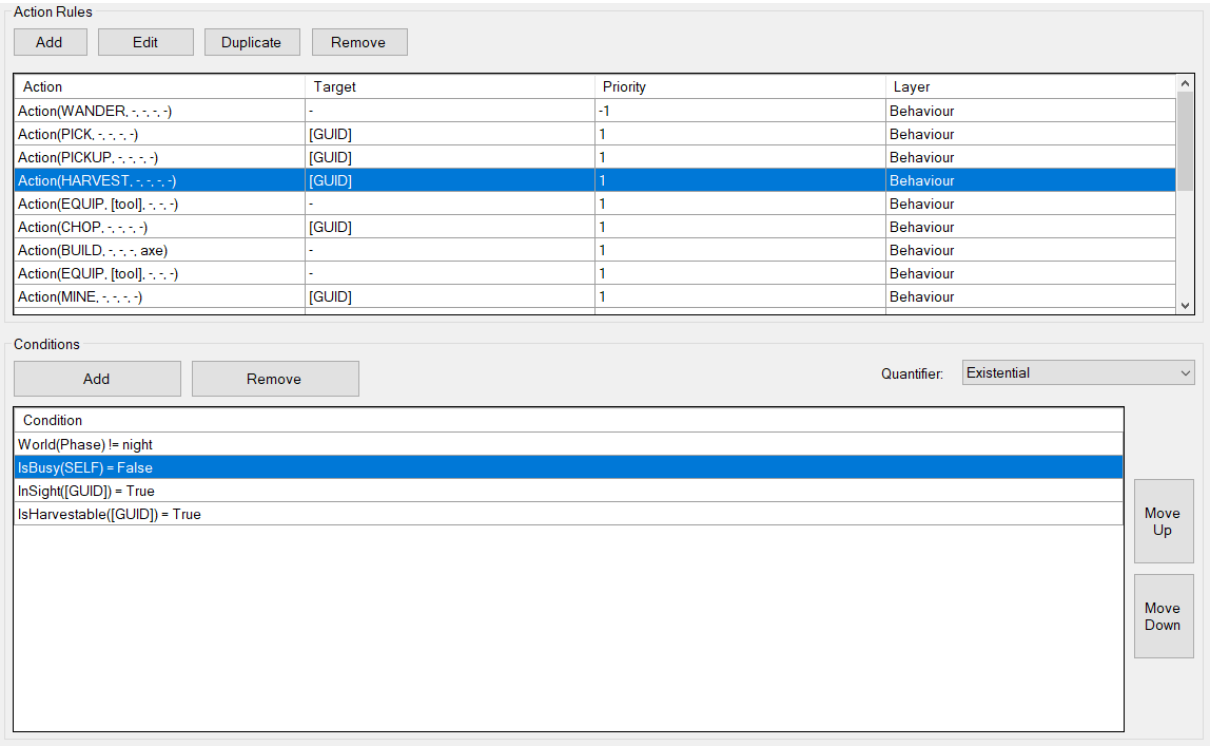

Figure 5.3: Walter's `HARVEST` action.

allows for reactive behaviour and prevents the overlap of actions with equal priority.

The developer should use this belief as a condition for actions, as shown in Figure 5.3, and omit this condition for actions that should be executed even if another one is being executed, as shown in Figure 5.5. In this example, equipping the torch can overlap other actions as it represents a case of life and death for the NPC (being in the dark will get the NPC dead).

`InLight`, `InSight`, and `InInventory` are in fact beliefs about entities. However, they are presented here as they represent the state of entities in relation to the NPC. For example, Walter will only try to harvest any given entity if it is in sight, as shown in Figure 5.3. The use of these three beliefs is highly recommended when the action requires interaction of the NPC with other entities. The NPC should not try to directly pick or pick up entities that she cannot see as they may no longer exist.

Do note that most of these beliefs require the agent's name. While defining the Decision Making Asset rules, the special keyword `SELF` should be used for this purpose. In Figure 5.4 we can see the NPC executing an action with some debug information being displayed in an overlay.

As far as world state beliefs go (described in Table 5.2), these are used to represent both the state of the world and entities present in the world.

Most beliefs used to describe entities coincide with the presence of certain *components* (as described in 5.1.2). The use of these beliefs as conditions for the appropriate actions is highly encouraged as they will prevent the NPC from trying to execute actions she cannot execute.

Figure 5.4: Walter executing a `PICK` action with debug information being displayed in an overlay.

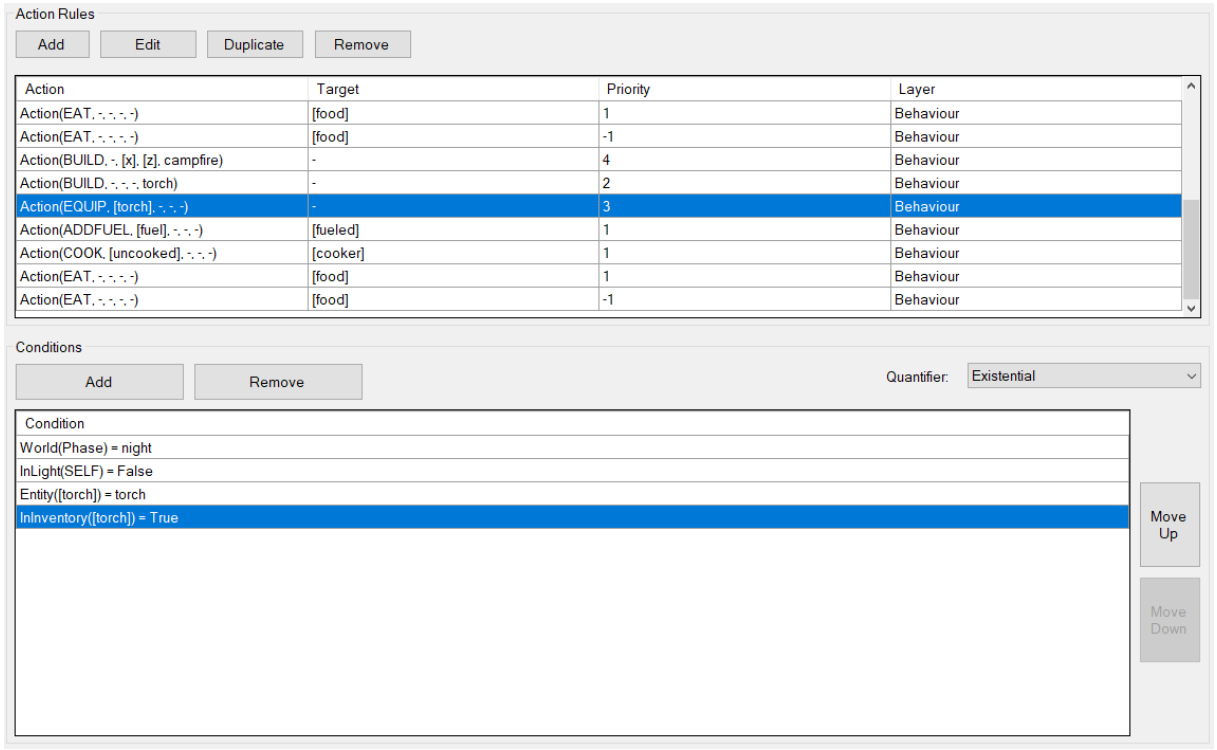

Figure 5.5: Walter's `EQUIP` torch example.

Figure 5.6: Example of speak action.

Walter will only harvest resources if the `IsHarvestable` belief is true, as depicted in Figure 5.3. This is also true for every other action present in Walter's decision rules.

Despite this, one can use the prefab of an entity as a condition, such as Walter does when equipping a torch (Figure 5.5). This will allow the creation of more refined behaviour.

Appendix A contains two tables which have a comprehensive list of all available actions, Tables A.1 and A.3. Both tables also contain a brief description of every action, which along side the existing example of Walter, will help developers write their own behaviour rules.

5.4.1 Dialogue

As far as speech goes, the current framework has some limitations. Currently, there is no two-way communication, meaning that, although the agent can speak (through the use of text), the players will not be able to respond. In Figure 5.6 we can see the execution of a speak action by the agent while he is executing another action.

To make an NPC speak, the developer will need to create the available utterances using the Dialogue Editor as depicted in Figure 5.7. Multiple utterances can be created with the same meaning, allowing for variety in the NPC's dialogue. In Walter's example, several utterances have been added for each meaning.

To trigger these sentences, the developer also needs to add a rule with the appropriate layer to the decision making asset, as shown in Figure 5.8. Here we can see, that in order for Walter to say a comment about hunger, his `Hunger` belief has to have a value lower than seventy.

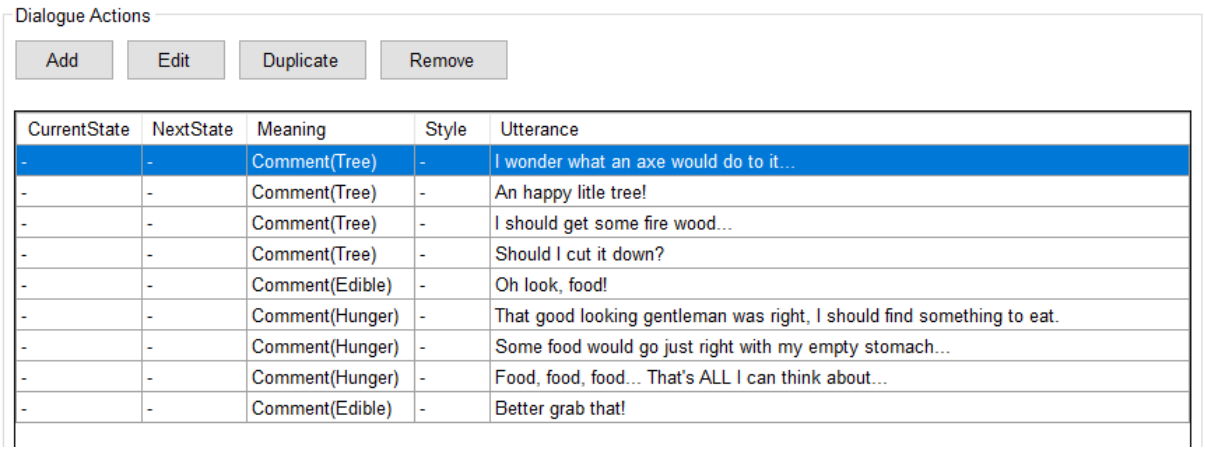

Figure 5.7: Walter's available dialogues.

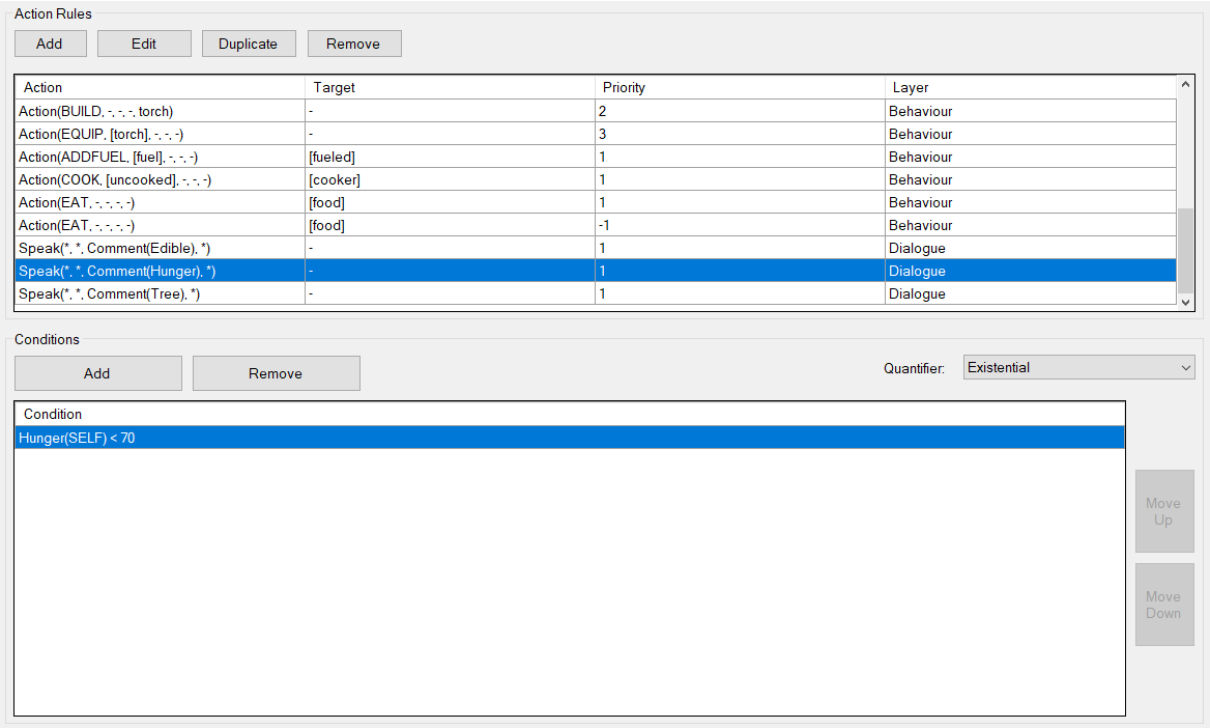

Figure 5.8: Walter's dialogue action rule.

Figure 5.9: Configuration screen for FAtiMA-DST.

The addition of these sentences allows players to better understand the NPC's decision process. By stating some random utterances about its current state, we can transmit the characters state of mind to the players. These sentences can also be used to transmit the NPC emotional state, needs, and goals.

5.5 Playing with Walter

In order to run our framework, players will have to run FAtiMA-Server by launching the console application available at the public repository. Then, before launching a game, the player will need to activate and configure the *mod* as shown in Figure 5.9. When the game is launched, the number of characters specified will spawn in the same place that players spawn.

In Figure 5.10, we can see an example of a player playing the game with two NPCs being controlled by FAtiMA.

Figure 5.10: A player (center) playing the *mod* with two FAtiMA controlled characters (top and bottom).

Table 5.2: World's State beliefs

Belief	Description
`Entity([GUID]) = [prefab]`	Defines an entity
`Quantity([GUID]) = [quantity]`	Defines how big is the stack (*quantity*) of a given entity
`IsCollectable([GUID]) = [bool]`	True if the given entity is pickable (collect natural resources). *PICK* action
`IsCooker([GUID]) = [bool]`	True if the given entity can cook other entities. *COOK* action
`IsCookable([GUID]) = [bool]`	True if the given entity can be cooked. *COOK* action
`IsEdible([GUID]) = [true]`	True if the entity may be eaten by the curent character (it takes into account the character's diet). *EAT* action
`IsEquippable([GUID]) = [bool]`	True if the given entity may be equipped. *EQUIP* action
`IsFuel([GUID]) = [bool]`	True if the given entity may be used to fuel stuff. *FUEL* action
`IsFueled([GUID]) = [bool]`	True if the given entity requires fuel to function. *FUEL* action
`IsGrower([GUID]) = [bool]`	True if the given entity can be used to grow seeds. *PLANT* action
`IsHarvestable([GUID]) = [bool]`	True if the given entity is ready to be harvested. *HARVEST* action
`IsPickable([GUID]) = [bool]`	True if the given entity is pickable (pick stuff from the ground). *PICKUP* action
`IsStewer([GUID])= [bool]`	True if the given entity can take other entities to cook recipes
`IsChoppable([GUID]) = [bool]`	True if the given entity is workable by an axe. *CHOP* action
`IsDiggable([GUID]) = [bool]`	True if the given entity is workable by a shovel. *DIG* action
`IsHammerable([GUID]) = [bool]`	True if the given entity is workable by a hammer. *HAMMER* action
`IsMineable([GUID]) = [bool]`	True if the given entity is workable by a pick. *MINE* action
`PosX([GUID]) = [value]`	Defines the X coordinate (*value*) of an entity
`PosZ([GUID]) = [value]`	Defines the Z coordinate (*value*) of an entity
`World(CurrentSegment) = [value]`	The current segment, ranges between 0 and 15
`World(Cycle) = [value]`	Defines how many cycles (days) have passed since the start of the game
`World(Phase) = [value]`	Defines the phase of the day. *value* can be: 'day', 'dusk', or 'night'
`World(PhaseLenght, [phase]) = [value]`	The current duration of the day *phase* in clock segments. The sum of all segments is always 16
`World(Season) = [value]`	Defines the current season. *value* can be: 'spring', 'summer', 'autumn', or 'winter'
`World(SeasonProgress) = [value]`	A value between 0 and 1 that defines the progress of the season
`World(ElapsedDaysInSeason) = [value]`	How many days have passed in the current season
`World(RemainingDaysInSeason) = [value]`	How many days are left to the end of the season
`World(SpringLength) = [value]`	Defines the current lenght of Spring
`World(SummerLength) = [value]`	Defines the current lenght of Summer
`World(AutumnLenght) = [value]`	Defines the current lenght of Autumn
`World(WinterLenght) = [value]`	Defines the current lenght of Winter
`World(IsSnowing) = [bool]`	True if it is snowing
`World(IsRaining) = [bool]`	True if it is raining
`World(MoonPhase) = [value]`	Defines the current moon phase. *value* can be: 'new', 'quarter', 'half', 'threequarter', or 'full'

6

Evaluation

Contents

By the end of this project we've successfully implemented a framework that enables developers to use an agency model, FAtiMA, to create NPCs for Don't Starve Together. This was achieved through the implementation of a modification for DST and a companion console application. As contributions resultant from this work we also count with a tutorial for creating NPCs and an example NPC.

For the remainder of this chapter we will talk about the published *mod*, then we will talk about a use case of the framework, and will finish with a comparison between our example character and the work of an anonymous developer.

6.1 Walter - The AI Companion

As said before, the example character we created to demonstrate the framework has been published in the Steam Workshop. At the time of writing, with 32 days of existence, the *mod* counts with 699 unique viewers and 118 subscribers, as shown in Figure 6.1.

While subscribers represent the number of people who've added the *mod* to their game, it does not tells us how many of those have actually used the *mod*. The *mod* also counts with little over 20 comments on the public page, mostly related with how to run the mod.

Although it has received no direct negative feedback, we believe that the necessity of running a separate console application, has greatly impacted the community's acceptance and use of the *mod*. Additionally, to the best of our knowledge, no member of the community has used our framework to develop their own NPCs.

6.2 Monte Carlo Tree Search Project

As part of the Artificial Intelligence for Games course taught at Instituto Superior Técnico, a group of three students used our framework for their final project. The project consisted in the implementation of the Monte Carlo Tree Search (MCTS) algorithm for the decision process of the agent.

Briefly, the MCTS algorithm is based in a random exploration of a state space. By randomly (and rapidly) collecting data of the space state, the algorithm will gradually improve its knowledge of what the best next move is.

Unlike many other algorithms, MCTS does not run until an answer is found. Instead, the usual approach consists in letting the algorithm run for some time (or a certain number of iterations) and then return the best possible action, in regard to the information collected so far. The algorithm has proven to be especially good in problems where the space state has an high branching factor and many possible combinations, such as the Go game.

In this case, the implementation of the MCTS algorithm was achieved through the use of FAtiMA's

Figure 6.1: Walter - The AI Companion *mod* lifetime stats.

Dynamic Properties. The created Dynamic Property, called `MCTS`, will use current knowledge base and apply the MCTS algorithm. The MCTS algorithm was let run for two thousand iterations (about twenty seconds) before returning a plan to act upon.

One of the main difficulties found during this use case was related to the random exploration of the state space. As Don't Starve Together does not provide a way to simulate the resulting state of applying actions and FAtiMA did not have such functionality either (although it can now achieve this through the use of the World Model Asset), the group had to create their own simulator of the Don't Starve Together world.

This work, praises the versatility of our framework by demonstrating the ability to create a planning agent based on MCTS. Although the group had to struggle with the world simulation, the end result was an NPC for DST with planning capabilities.

6.3 Walter vs. Artificial Wilson

As a means of evaluating the created NPC, we've run it and recorded how long it can survive. Additionally, we compare it to an unpublished character for Don't Starve, Artificial Wilson.

Artificial Wilson, is the work of the anonymous *modder* KingofTown and can be found in the public repository https://github.com/KingofTown/DS-AI.

6.3.1 Testing Conditions

For each run of the characters, a new world was dynamically generated with equal sets of constraints for both characters. The characters where then let run for as long as possible until their death.

As we said, Artificial Wilson was created for the original game, Don't Starve. Therefore it is incompatible with Don't Starve Together because of the updates made to turn it into a multiplayer game. However, the game mechanics and configurations remain the same.

As such, we can use the scenario configurator to generate equivalent worlds. For this test, we considered the following restrains:

- **A world with no enemies**: all aggressive monsters have been removed from the game as well as the giants.
- **No random meteorological events**: apart from raining all other events have been disabled (like meteors, frog rains, lighting storms, and wildfires).
- **No resurrection stones**: if the NPC dies, it stays dead.
- **No matting season for *beefalos***: during this season these, normally pacific creatures, will attack any character on sight.
- **No modifications**: apart from the necessary ones for running both characters.
- **Only one NPC and no players**: while in Don't Starve this does not apply, in Don't Starve Together the tests where run locally with no players present in the world.

These constrains will generate similar worlds for both Don't Starve and Don't Stave Together. For each play-through of an agent, a new world is generated with these sets of constraints. Each generated world is unique but complies to the specified set of constraints.

6.3.2 The Results

Each character has been run thirty times and the survival duration for each character is presented in Figures 6.2 and 6.3.

Walter survived, on average, one point eight days while Artificial Wilson survived, on average, eight point nine days. The percent difference of 132% between both values can be explained by the detail put in each character's decision process.

Walter is a model based agent with nineteen rules of decision. Each of these rules has no decision process besides the character's ability to perform it (e.g. does the NPC has an axe equipped to cut down a tree?, does it have the required ingredients to build a fire?).

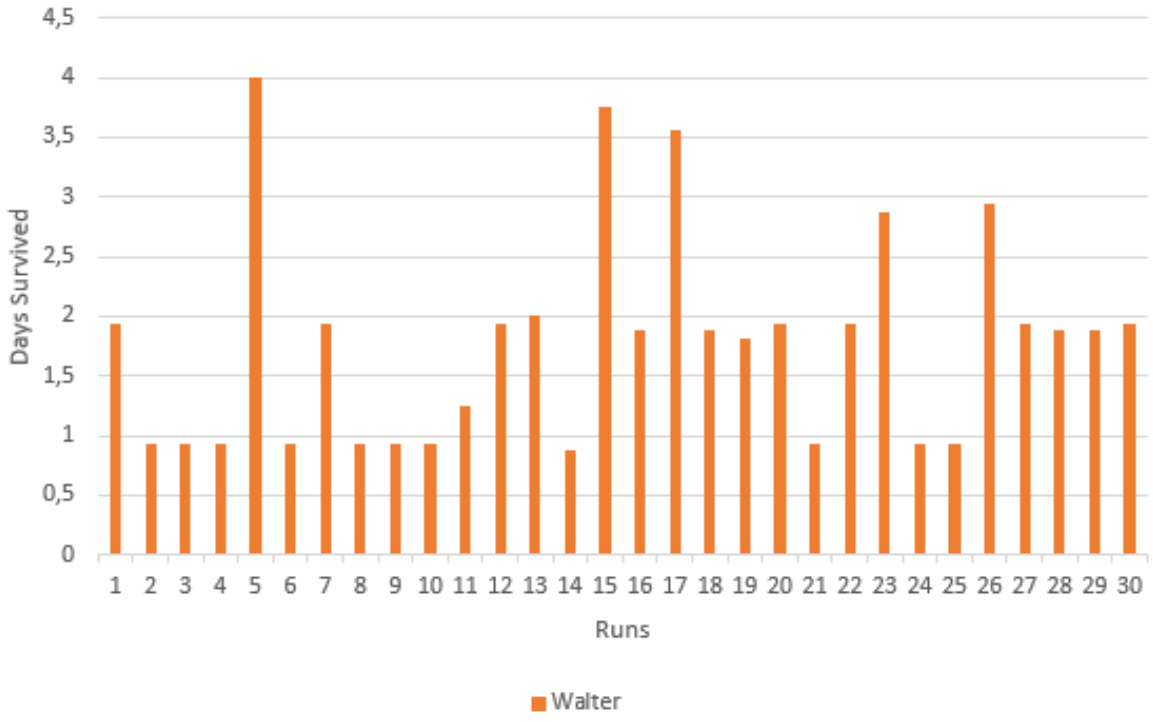

Figure 6.2: Days survived for Walter.

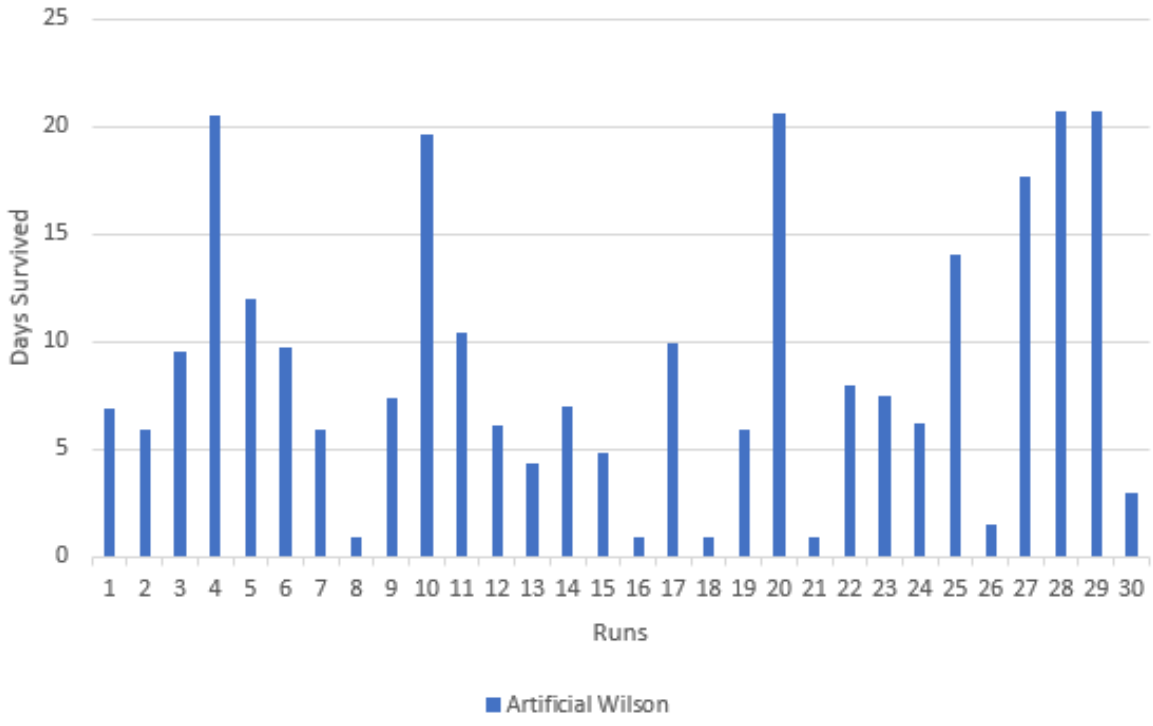

Figure 6.3: Days survived for Artificial Wilson.

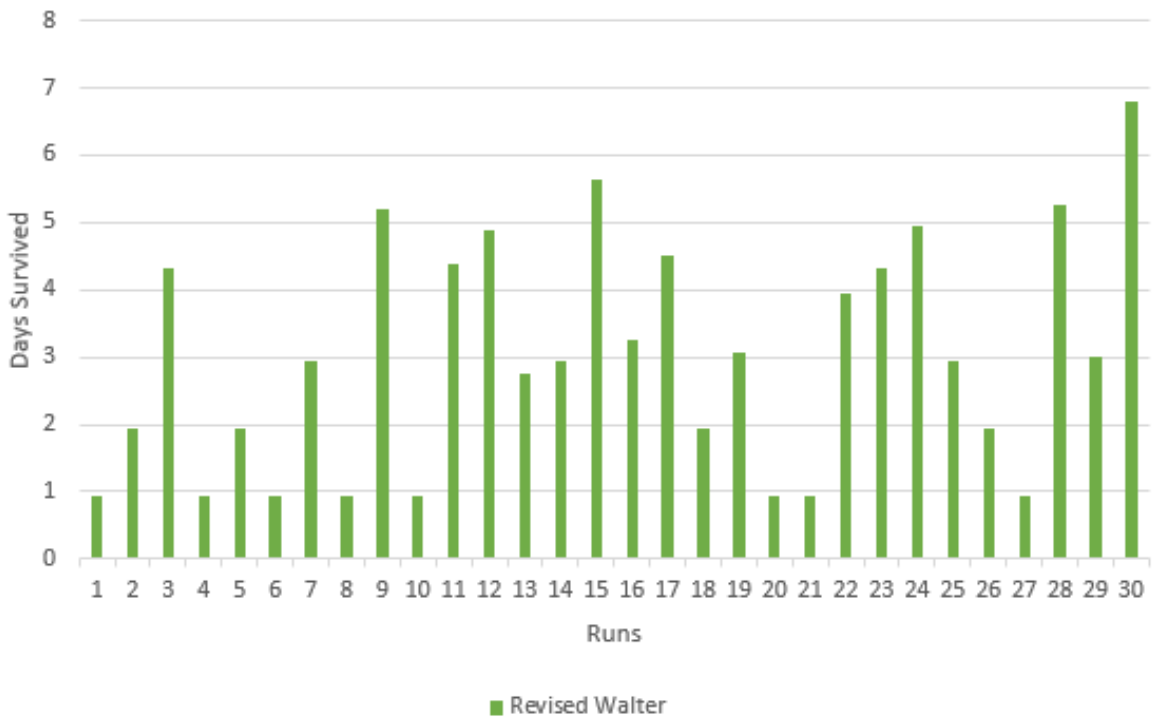

Figure 6.4: Days survived for revised version of Walter.

Artifical Wilson is based on the built-in behaviour trees. The final behaviour tree counts with over seventy nodes and dozens of helper functions. Embedded in the NPCs's decision process are some planning capabilities and resource management (e.g. the NPC will ignore resources if she already has gathered enough of it), which Walter does not possess.

If we compare both characters to real human players, Walter has the ability of a newcomer who as never played the game, while Artificial Wilson could be compared to a returning player that played the game for a couple dozen of times[1].

For each character we've recorded not only their survival time but also their cause of death. In Table 6.1 you can see a summary for both characters.

Table 6.1: Artificial Wilson, Walter, and Revised Walter cause of death summary.

Artificial Wilson		Walter		Revised Walter	
Cause of Death	Times	Cause of Death	Times	Cause of Death	Times
Cold	5	Darkness	18	Darkness	17
Cactus	2	Fire	8	Hunger	13
Darkness	10	Hunger	4		
Hunger	13				

We can see that Walter died several times due to fire damage. Upon analyzing its behaviour, we

[1] This comparison has been based on the writer's experience with the game

observed that this was due to the NPC's placement of campfires. The reactive rules we've empowered Walter with, have no concern for the placement of structures. As such, the NPC would often build campfires in the middle of forests that would be set ablaze by the campfire. This, in turn, would cause the NPC to die by fire.

In order to counteract this, we decided to disable the rule for building campfires. Instead the agent relies on torches to illuminate itself through the night. The results are presented in Figure 6.4. The average surviving time for Walter increased from one point eight days to three days, which represents an increase of 66%.

6.3.3 Notes on Character Behaviour

We would like to note some aspects we found interesting from both character's behaviour.

While Artificial Wilson has a selective picking and manages his inventory, Walter's reactive behaviour lacks this features. In fact, for runs that Walter was able to survive more that two, three days it became apparent that the lack of space in the inventory was a great hindrance. Most of the time Walter would collect many items that would just occupy space in the inventory without ever using them.

Regarding the map exploration, Artificial Wilson would end up by exploring a lot more of the map than Walter. This is directly related to the point discussed before. By picking only what it needs, Artificial Wilson would move on to unexplored territories to find what it needed. On the other hand, Walter would stay around the same place for extensive amounts of time.

Which brings us to the third aspect we would like to discuss. By staying on the same area, Walter would eventually deplete all the available resources. Which in fact killed him most of the time, due to the lack of food or materials to make a torch for the night. Artificial Wilson however, did not suffer from this problem has it roamed the world more extensively, being able to recollect certain resources that would regrow while he was exploring other parts of the world.

7

Conclusion

Contents

In this dissertation we've argued how in video games that are heavily dependent on interactions with NPCs, many of them are not socially aware and behave in a non-believable manner.

Firstly, we have looked at how the video game industry is currently making NPCs and concluded that the used techniques do not involve current frameworks and architectures for social agents. The use of Behaviour Trees and Finite State Machines, the most commonly used techniques in today's industry, provide skilled game designers with the ability to create somewhat complex behaviour. However, these techniques can quickly become an authoring burden. Having to manage hundreds of behaviour nodes and or states is no simple task. Furthermore, grasping the context of social interaction will often be left out of the NPC's behaviour.

Focusing on survival games, we've noticed how these games will often lack the presence of NPCs, and when they are present, they can break the player's immersion by not being subject to the same rules as the players. This, allied to the fact that many survival games rely on multi-player interaction in order to keep interesting, motivated us to create a platform that allows developers to create believable NPCs with interesting behaviour for survival games.

To achieve this, we had to explore what other platforms are currently available to develop NPCs. To the best of our knowledge there are no publicly available platforms that allow developers to create NPCs for survival games. We've then decided to look into agent creation platforms and explored tools like JADE, NetLogo, and PsychSim. Although they represent good options for agent engineering and simulations they do not provide us with the necessary tools to solve our problem, the creation of believable NPCs in survival games.

In order to make believable NPCs we need them to possess certain human-like traits: social awareness, active goal pursuit, and reactivity, for example. As such, the next step was to look at agency models that concern themselves with believability and social interaction without forfeiting reasoning and planning. Versu, CiF, and FAtiMA, are all fully-fledged agency models, but we decided to use FAtiMA due to its versatility and decision making capabilities. By incorporating FAtiMA's capabilities into a commercial survival game we've been able to realize a platform that allows developers to create NPCs with social awareness and planning capabilities.

We've chosen Don't Starve Together for its multiplayer nature, it's ability to configure the world we play in, the possibility of including our own modifications to the game natively, and for its sanity mechanic. A novelty among its peers, the sanity mechanic represents the character's mental health, another component that enhances the player's immersion by giving the character a human trait.

Then we implemented the *mod*, FAtiMA-DST, and the console application, FAtiMA-Server, that allows us to run AI powered NPCs in DST. As an example and proof of concept, we've also created Walter, a model based NPC that was published in the Steam Workshop.

Additionally, the platform has been successfully used to implement a planning agent using the MCTS

algorithm. By creating a new Dynamic Property, the developers where able to create an agent using FAtiMA and run it in DST.

In order to evaluate our work, we've compared our example agent, Walter, with Artificial Wilson, an agent created using the in-game's behaviour trees by an anonymous *modder*. Although Walter's performance is poor when compared to Artificial Wilson, the discrepancy can be justified due to the difference in the size of both implementations. While Walter counts with nineteen rules of decision, Artificial Wilson counts over seventy nodes in its behaviour tree.

This dissertation provides the ground work for future developments in the creation of NPCs for survival games, in particular for Don't Starve Together. It is our belief that with an equal effort, Walter's behaviour would at least match the performance of Artificial Wilson with the added value of social awareness, emotional responses, and active goal pursuit.

7.1 Future Work

As this work stands, it allows the creation of NPCs for Don't Starve Together. However, the platform has great room for improvement.

- Two-way communication. Currently, the platform allows the NPC to talk and make remarks on the world. However, the player is not able to respond and talk back to the NPC. Either by allowing chat responses or by adding an additional component to the game's interface, the interaction with the NPC could be greatly improved.
- Add more beliefs. Expanding the character's available beliefs will enable us to create more complex behaviour. However, only by adding behaviour to NPCs, will we understand which beliefs are effectively necessary.
- Improve Walter. As a proof of concept Walter plays its role as expected. Nonetheless, Walter is a simple model based agent that does not make full use of FAtiMA's capabilities. Although the framework is fully prepared to support FAtiMA, the character does not benefit from all the capabilities FAtiMA might grant to a virtual agent. By incorporating additional assets into the RPC's definition, like the Emotional Appraisal Asset or the Social Importance Asst, Walter could be further improved.
- Reduce the burden of running FAtiMA's characters in DST. As it stands, players have to run a console application in parallel with the game for the platform to work properly. It is our belief that by removing this requirement, more players would be able to play with NPCs that make use of our platform. Ideally, this would be done by incorporating FAtiMA's DLLs into the embedded Lua interpreter that DST comes with. However, this would require some sort of collaboration with the

game developers, Klei Entertainment. Alternatively, a public server could be set up to host the FAtiMA-Server component.

- Add an additional layer of decision. Players can express feelings through the use of gestures. Gestures are animations that represent different states of mind the player may be feeling (some of the available gestures are ”annoyed”, ”bye”, ”angry”, ”joy”, ”dance”, and ”sad”). The addition of this capability to the platform would be a great improvement as far as believability goes for created NPCs.
- Add more composed actions to the platform. Currently the only composed behaviour incorporated in the platform is the wander behaviour. The addition of combat behaviour, utilization of chests, structure placement helpers, and others, would greatly improve an NPC's capabilities.
- Different NPCs running different RPCs. For each NPC there should be an ability to specify the RPC to be loaded. This would enable the creation of richer scenarios by using characters with different characteristics, goals, reactions, dialogues, among others.

Bibliography

[1] E. Brown and P. Cairns, "A grounded investigation of game immersion," in *CHI'04 extended abstracts on Human factors in computing systems*. ACM, 2004, pp. 1297–1300.

[2] P. R. Thrainsson, A. L. Petursson, and H. H. Vilhjálmsson, *Dynamic Planning for Agents in Games Using Social Norms and Emotions*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 473–474.

[3] W. IJsselsteijn, Y. de Kort, K. Poels, A. Jurgelionis, and F. Bellotti, "Characterising and measuring user experiences in digital games," pp. 1–4, 2007.

[4] J. Bates *et al.*, "The role of emotion in believable agents," *Communications of the ACM*, vol. 37, no. 7, pp. 122–125, 1994.

[5] K. Entertainment, "Don't starve," Android, iOS, Linux, Microsoft Windows, OS X, PlayStation 3, PlayStation 4, PlayStation Vita, Wii U, Xbox One, Nintendo Switch, 2013.

[6] ——, "Don't starve together," PlayStation 4, Microsoft Windows, Linux, Xbox One, macOS, 2016.

[7] M. Guimarães, "Skyrim Mod for Social NPCs," Master's thesis, Instituto Superior Técnico, Portugal, 2016.

[8] N. Afonso and R. Prada, "Agents that relate: Improving the social believability of non-player characters in role-playing games," in *ICEC'2008 - 7th International Conference on Entertainment Computing*, ser. Lecture Notes in Computer Science, vol. 5309/2009. Pittsburgh, PA, USA: Springer Berlin / Heidelberg, September 2008, pp. 34–45.

[9] M. O. Ferreira, "A Merchant Model Improving player experience with NPC vendors in Role Playing Games," Master's thesis, Instituto Superior Técnico, Portugal, 2017.

[10] G. N. Yannakakis, "Game ai revisited," in *Proceedings of the 9th Conference on Computing Frontiers*, ser. CF '12. ACM, 2012, pp. 285–292.

[11] G. Games, "Horizon zero dawn," Playstation 4, 2017.

[12] Thomas, *Disney Animation: The Illusion of Life*. New York: Abbevillee Press, 1981.

[13] U. Montreal, "Assassin's creed unity," PC, PlayStation 4, Xbox One, 2014.

[14] B. Edmonton, "Dragon age: Inquisition," PC, PlayStation 3, Xbox 360, PlayStation 4, Xbox One, 2014.

[15] B. G. Studios, "The elder scrolls v: Skyrim," PC, PlayStation 3, Xbox 360, PlayStation 4, Xbox One, Nintendo Switch, 2011.

[16] M. Guimarães, P. Santos, and A. Jhala, "Cif-ck: An architecture for social npcs in commercial games," *CiG 2017, New York, USA*, 2017.

[17] R. Evans and E. Short, "Versu — a simulationist storytelling system," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 6, no. 2, pp. 113–130, 2013.

[18] J. McCoy, M. Treanor, B. Samuel, M. Mateas, and N. Wardrip-Fruin, "Prom week: social physics as gameplay," in *Proceedings of the 6th International Conference on Foundations of Digital Games*, ACM. ACM, 2011.

[19] A. Sullivan, A. Grow, M. Mateas, and N. Wardrip-Fruin, "The design of mismanor: creating a playable quest-based story game," in *Proceedings of the International Conference on the Foundations of Digital Games*, ACM. ACM, 2012, p. 180–187.

[20] M. Mateas and A. Stern, "Façade: An experiment in building a fully-realized interactive drama," in *Game developers conference*, vol. 2, 2003, pp. 4–8.

[21] F. Bellifemine, A. Poggi, and G. Rimassa, "Jade–a fipa-compliant agent framework," in *Proceedings of PAAM*, vol. 99, no. 97-108. London, 1999, p. 33.

[22] P. D. O'Brien and R. C. Nicol, "Fipa — towards a standard for software agents," *BT Technology Journal*, vol. 16, no. 3, pp. 51–59, Jul 1998. [Online]. Available: https://doi.org/10.1023/A:1009621729979

[23] S. Tisue and U. Wilensky, "Netlogo: A simple environment for modeling complexity," in *International conference on complex systems*, vol. 21. Boston, MA, 2004, pp. 16–21.

[24] S. C. Marsella, D. V. Pynadath, and S. J. Read, "Psychsim: Agent-based modelling of social interactions ans influence," in *Proceedings of the International Conference on Cognitive Modeling*, 2004, pp. 243–248.

[25] A. Whiten, *Natural theories of mind : evolution, development, and simulation of everyday mindreading / edited by Andrew Whiten*. B. Blackwell Oxford, UK, 1991.

[26] D. V. Pynadath, M. Si, and S. C. Marsella, “Modeling Theory of Mind and Cognitive Appraisal with Decision-Theoretic Agents,” in *Appraisal*, Apr. 2011, pp. 1–30.

[27] J. McCoy, M. Treanor, B. Samuel, A. A. Reed, M. Mateas, and N. Wardrip-Fruin, “Social story worlds with comme il faut,” *IEEE Transactions On Computational Intelligence and AI in Games*, vol. 6, 2014.

[28] J. McCoy, M. Treanor, B. Samuel, N. Wardrip-Fruin, and M. Mateas, “Comme il faut: A system for authoring playable social models,” in *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE’11. AAAI Press, 2011, pp. 158–163.

[29] M. Treanor, J. McCoy, and A. Sullivan, “A framework for playable social dialogue,” in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[30] R. Evans, *Introducing Exclusion Logic as a Deontic Logic*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 179–195.

[31] J. Dias, S. Mascarenhas, and A. Paiva, *FAtiMA Modular: Towards an Agent Architecture with a Generic Appraisal Framework*. Cham: Springer International Publishing, 2014, pp. 44–56.

[32] R. S. Aylett, S. Louchart, J. Dias, A. Paiva, and M. Vala, “Fearnot!–an experiment in emergent narrative,” in *International Workshop on Intelligent Virtual Agents*. Springer, 2005, pp. 305–316.

[33] A. Paiva, J. Dias, D. Sobral, R. Aylett, S. Woods, L. Hall, and C. Zoll, “Learning by feeling: Evoking empathy with synthetic characters,” *Applied Artificial Intelligence*, vol. 19, no. 3, pp. 235–266, 2005.

[34] S. H. Rodrigues, S. Mascarenhas, J. M. A. Dias, and A. Paiva, “I can feel it too! emergent empathic reactions between synthetic characters,” in *Affective Computing and Intelligent Interaction and Workshops, 2009. ACII 2009. 3rd International Conference on*, no. 10.1109/ACII.2009.5349570. IEEE, 2009, pp. 1 –7.

[35] R. Aylett, N. Vannini, A. Paiva, S. Enz, and L. E. Hall, “But that was in another country: agents and intercultural empathy,” in *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, vol. 1. IFAAMAS, 2009, pp. 329–336.

[36] F. Correia, P. Alves-Oliveira, N. Maia, T. Ribeiro, S. Petisca, F. S. Melo, and A. Paiva, “Just follow the suit! trust in human-robot interactions during card game playing,” *IEEE RO-MAN 2016, New York, USA.*, 2016.

[37] I. J. Roseman and C. A. Smith, “Appraisal theory: Overview, assumptions, varieties, controversies,” in *Appraisal processes in emotion: Theory, methods, research*, A. Schorr and K. R. Scherer, Eds. London: Oxford University Press, 2001, pp. 221–232.

[38] A. Ortony, G. L. Clore, and A. Collins, *The cognitive structure of emotions*. Cambridge, New York: Cambridge Univ. Press, 1988.

[39] T. D. Kemper, *Power and Status and the Power-Status Theory of Emotions*. Boston, MA: Springer US, 2006, pp. 87–113. [Online]. Available: https://doi.org/10.1007/978-0-387-30715-2_5

[40] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall Press, 2009.

List of Actions

Table A.1: The first part of the available actions.

Actions	Restrictions	Description
Action(ACTIVATE, -, -, -, -) = [target]	target: GUID	Interact with some game elements
Action(ADDFUEL, [invobject], -, -, -) = [target]	invobject: GUID, target: GUID	Add fuel to fueled entities (campfire, firesupressor)
Action(ATTACK, -, -, -, -) = [target]	target: GUID	Attack other entities
Action(BAIT, [invobject], -, -, -) = [target]	invobject: GUID, target: GUID	Put bait on traps
Action(BUILD, -, [x], [z], [recipe]) =	([x], [z]): position, recipe: recipe's name	Depending on weather you are crafting an item or placing a structure you'll need to pass a value to the *(x, z)* parameters
Action(CASTSPELL, [invobject], -, -, -) = [target]	invobject:GUID, target: GUID	Cast magic item at *target*. If *invobject* is not specified, the equipped item is used.
Action(CHECKTRAP, -, -, -, -) = [target]	target: GUID	Check if the given trap has caught anything
Action(CHOP, -, -, -, -) = [target]	target: GUID	Chop trees, an axe must be equipped in order to use
Action(COMBINESTACK, [invobject], -, -, -) = [target]	invobject: GUID, target: GUID	Combines the given *invobject* into *target* if it is the same prefab and target is not full
Action(COOK, [invobject], -, -, -) = [target]	invobject: GUID, target: GUID	Cook *invobject* at the specified *target*
Action(DEPLOY, [invobject], [x], [z], -) = -	invobject: GUID, ([x], [z]): position	Place ground tile, walls, fences, and gates
Action(DIG, -, -, -, -) = [target]	target: GUID	Dig grass, twigs, rabbit holes, graves, and others from the ground
Action(DROP, [invobject], [x], [Z], -) = -	invobject: GUID, ([x], [z]): position	Drop held item to a spot in the ground
Action(DRY, [invobject], -, -, -) = [target]	invobject: GUID, target: GUID	Dry meat at racks
Action(EAT, -, -, -, -) = [target]	target: GUID	Eat food
Action(EQUIP, [invobject], -, -, -) = -	invobject: GUID	Equip an item that is in the character's inventory
Action(EXTINGUISH, [invobject], -, -, -) = [target]	invobject: GUID, target: GUID	Use the *invobject* to extinguish the burning *target*
Action(FEED, [invobject], -, -, -) = [target]	invobject: GUID, target: GUID	Feed the *invobject* to the *target*
Action(FEEDPLAYER, [invobject], -, -, -) = [target]	invobject: GUID, target: GUID	Feed the player (*target*) with *invobject* (might work the same has the above)
Action(FERTILIZE, [invobject], -, -, -) = [target]	invobject: GUID, target: GUID	Use *invobject* to Fertilize the *target*
Action(FILL, [invobject], -, -, -) = [target]	invobject: GUID, target:GUID	Fill the mosquito sack (*invobject*) at a pond (*target*)
Action(FISH, -, -, -, -) = [target]	target: GUID	Use a fishing rod (must be equipped) to fish in a pond (*target*)
Action(GIVE, [invobject], -, -, -) = [target]	invobject: GUID, target: GUID	Give *invobject* to *target*
Action(GIVEALLTOPLAYER, [invobject], -, -, -) = [target]	invobject: GUID, target: GUID	Give all of *invobject* to player (*target*)
Action(GIVETOPLAYER, [invobject], -, -, -) = [target]	invobject: GUID, target: GUID	Give *invobject* to player (*target*) (Not sure on the difference of these three actions)
Action(HAMMER, -, -, -, -) = [target]	target: GUID	Hammer down built structures (*target*)
Action(HARVEST, -, -, -, -) = [target]	target: GUID	Harvest crops and cookpots
Action(HEAL, [invobject], -, -, -) = [target]	invobject: GUID, target: GUID	Use *invobject* to heal the *target*

Table A.3: The second part of the available actions.

Actions	Restrictions	Description
`Action(JUMPIN, -, -, -, -) = [target]`	`target: GUID`	Jump into wormhole (*target*)
`Action(LIGHT, -, -, -, -) = [target]`	`target: GUID`	Set the *target* on fire (must have a torch equipped)
`Action(LOOKAT, -, -, -, -) = [target]`	`target: GUID`	Face the *target*
`Action(MANUALEXTINGUISH, -, -, -, -) = [target]`	`target: GUID`	Use your hands to try and extinguish fires
`Action(MINE, -, -, -, -) = [target]`	`target: GUID`	Mine rocks, sinkholes, glassiers, etc (must have a pickaxe equipped)
`Action(MOUNT, -, -, -, -) = [target]`	`target: GUID`	Mount a saddled mount (*target*)
`Action(MURDER, -, -, -, -) = [target]`	`target: GUID`	Murder targeted inocent creature (e.g. rabbits) while in inventory
`Action(NET, -, -, -, -) = [target]`	`target: GUID`	Use nets to catch bugs (*target*)
`Action(PICK, -, -, -, -) = [target]`	`target: GUID`	Pick the targeted resource (e.g. grass, saplings, berry bushes, etc)
`Action(PICKUP, -, -, -, -) = [target]`	`target: GUID`	Pick up items from the ground (e.g. rocks, twigs, cutgrass, etc.)
`Action(PLANT, [invobject], -, -, -) = [target]`	`invobject: GUID, target: GUID`	Plant *invobject* (seeds) into *target*
`Action(REEL, -, -, -, -) = [target]`	`target: GUID`	Reel in the fish while fishing (the target is the pond)
`Action(RESETMINE, -, -, -, -) = [target]`	`target: GUID`	Reset mines like the tooth trap
`Action(RUMMAGE, -, -, -, -) = [target]`	`target: GUID`	Rummage about in a container
`Action(SADDLE, [invobject], -, -, -) = [target]`	`invobject: GUID, target: GUID`	Use *invobject* to saddle up the *target*
`Action(SEW, [invobject], -, -, -) = [target]`	`invobject: GUID, target: GUID`	Use *invobject* to sew the *target*
`Action(SHAVE, [invobject], -, -, -) = [target]`	`invobject: GUID, target: GUID`	Use the *invobject* to shave the *target*
`Action(SLEEPIN, -, -, -, -) = [target]`	`target: GUID`	Sleep in the *target* (tent or sleeping bag)
`Action(SMOTHER, -, -, -, -) = [target]`	`target: GUID`	Smother the smoking *target* (stuff about to burst into flames)
`Action(STORE, [invobject], -, -, -) = [target]`	`invobject: GUID, target: GUID`	Store the *invobject* into the *target*
`Action(TAKEITEM, [], -, -, -) = [target]`		—take brid from cage
`Action(TERRAFORM, [invobject], [x], [z], -) = -`	`invobject: GUID, ([x], [z]): position`	Use the *invobject* to terraform the *position*
`Action(TURNOFF, -, -, -, -) = [target]`	`target: GUID`	Turn the *target* off (e.g. firesupressor)
`Action(TURNON, -, -, -, -) = [target]`	`target: GUID`	Turn the *target* on
`Action(UNEQUIP, -, -, -, -) = [target]`	`target: GUID`	Unequip *target*
`Action(UNSADDLE, -, -, -, -) = [target]`	`target: GUID`	Remove the saddle from the *target*
`Action(UPGRADE, [invobject], -, -, -) = [target]`	`invobject: GUID, target: GUID`	Use *invobject to upgrade the *target* (e.g. upgrade a wall)
`Action(WANDER, -, -, -, -) = -`		This is a the behaviour of wandering about the world, not really an action
`Action(WALKTO, -, -, -, -) = [target]`	`target: GUID`	Walk up to the *target*