Goal-Oriented Hierarchical Task Networks and Its Application on Interactive Narrative Planning

by

Emir Artar

Submitted to the Graduate School of Engineering and Natural Sciences

in partial fulfillment of

the requirements for the degree of

Master of Science

Sabancı University

August 2019

Goal-Oriented Hierarchical Task Networks and Its Application on Interactive Narrative Planning

APPROVED BY:

Prof. Dr. Berrin Yanıkoğlu
(Thesis Supervisor)

Assoc. Prof. Dr. Barbaros Bostan
(Thesis Co-Advisor)

Prof. Dr. Albert Levi

Assoc. Prof. Dr. Hüsnü Yenigün

Asst. Prof. Dr. Reyyan Yeniterzi

DATE OF APPROVAL: 16/09/2019

Abstract

GOAL-ORIENTED HIERARCHICAL TASK NETWORKS AND ITS APPLICATION ON INTERACTIVE NARRATIVE PLANNING

Emir Artar

Computer Science and Engineering, Master Thesis, August 2019

Thesis Advisor

Prof. Berrin Yanikoglu

Thesis Co-Advisor

Assoc. Prof. Barbaros Bostan

Keywords: Artificial Intelligence, Game Design, Narrative Planning

Abstract

Two of the most commonly used AI architectures in digital games are Behavior Tree (BT) and Goal-Oriented Action Planning (GOAP). The BT architecture is script based, highly controllable but barely expandable. On the other hand the GOAP architecture is planner based, barely controllable but highly expandable. This thesis proposes a hybrid AI architecture called Goal-Oriented Hierarchical Task Network (GHTN); combining planner based approach of GOAP with script based approach of BT. GHTN modifies the Hierarchical Task Network (HTN) architecture by replacing its iterative planner with a goal oriented planner, while maintaining the BT-like scripting capabilities of HTN.

GHTN's iterative-planner hybrid architecture is suitable to be used for Interactive Narrative Planning. Using GHTN with a previously crafted domain, it is possible to obtain a non-repetitive and continuous narrative flow which can also be directed by external goals. The user is presented with choices that are intelligently chosen to push the narrative towards the goal; then, depending on the answers new choices are generated. The initial state of the world and the goals are specified by a Scenarist who has the knowledge of the domain. The proposed architecture is tested on Interactive Narrative Planning task with an example domain set in the Lala Land universe, and the architecture is tested with several initial world states and goals.

Özet

HEDEF ODAKLI HİYERARŞİK GÖREV AĞLARI
VE ETKİLEŞİMLİ ANLATIM PLANLAMADA UYGULAMASI

Emir Artar

Bilgisayar Bilimi ve Mühendisliği, Yüksek Lisans Tezi, Ağustos 2019

Tez Danışmanı

Prof. Berrin Yanikoğlu

Tez Eş Danışmanı

Assoc. Prof. Barbaros Bostan

Keywords: Yapay Zeka, Oyun Tasarımı, Anlatım Planlama

Özet

Dijital oyunların yapay zeka mimarilerinde en sık kullanılan yöntemler Karar Ağaçları (BT) ve Hedef-Odaklı Aksiyon Planlamadır (GOAP). Karar ağaçları mimarisi senaryo tabanlı çalıştığından ötürü çok kontrol edilebilirdir fakat genişletilmeye açık değildir. Bunun aksine GOAP mimarisi planlama temellidir, dolayısıyla kontrol edilebilirliği azdır fakat kolaylıkla genişletilebilirdir. Bu tez Hedef-Odaklı Hiyerarşik Görev Ağları'nı (GHTN) ileri sürer. GHTN; Planlama temelli olan Hedef-Odaklı Aksiyon Planlama mimarisi ile senaryo temelli Karar Ağacı mimarisinin karması olarak disayn edilmiştir. GHTN, Hiyeralşik Görev Ağları (HTN) yapısının mimarisinde değişikliklere giderek HTN'in yinelemeli planlama yapısını hedef odaklı bir planlama yapısı ile

değiştirir ve bu modifikasyon esnasında Karar Ağaçları'nda olduğu gibi bir senaryo yazım yapısını eklemeyi hedefler.

GHTN'nin senaryo-planlama karması mimarisi, Etkileşimli Anlatı Planlama için kullanılabilir. Öncesinde yaratılmış bir görev ağı ile beraber çalıştırıldığında, tekrarsız ve devamlı bir anlatı akışı sağlar ve bu anlatı akışının dışarıdan verilen hedefler çerçevesinde düzenler. Kullanıcıya, anlatıyı hedefe götürmek üzere akıllıca seçilmiş sorular sorulur ve kullanıcının yaptığı seçimler doğrultusunda hikayeyi hedefe doğru tekrar yönlendirir. Dünyanın başlangıç durumu ve hedefleri, görev ağına hakim bir Senarist tarafından seçilir. Bu tezde, sunulan GHTN mimarisne Etkileşimli Anlatı Planlama görevi verilmiştir. Anlatıda kullanılacak görev ağı, "Lala Land" dünyasından esinlenerek yaratılmış, ve çeşitli başlangıç ve hedef durumları ile sınanmıştır.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

AI	Artificial Intelligence
BT	Behavior Tree
GOAP	Goal Oriented Action Planning
HTN	Hierarchical Task Network
GHTN	Goal-Oriented Hierarchical Task Network

CHAPTER 1

Introduction

In modern video games both the game world and the action space available to the player is massive. Because of the unpredictable nature of the player, it is very costly if not impossible to design for every possible event chains in the game. Artificial Intelligence (AI) as a dynamic decision mechanism; provides a practical solution to this problem. Game AI's are designed to predict and counter player inputs. AI is a valuable tool used to enrich the experience and provide fair challenge to the player.

There are many aspects to a game that can be improved with the use of AI. Among these features most commonly known is AI controlled agents also known as bots. Bots can benefit or hinder the progress of player. By using a rubber-band AI approach, difficulty provided by these bots can change real-time, resulting in a challenging game independent from the skill of the player. AI can also be used as a hidden assistance tool, as in the use of AI in kinematic animations. In complex animations, predicting the player inputs result in fluent animations, a feat only possible because the use of AI. Another field that makes use of AI is Interactive Narrative, where systems can be designed to take on the role of a narrator, such as the popular culture examples of Black Mirror: Bandersnatch or dungeon masters in table top D&D games.

Repetitiveness and discontinuity must be avoided in Interactive Narrative. Stories where repetition happens often, or causality relationships are not established strongly appear unbelievable and artificial. Interactive Narrative planning is the arrangement of story content with the context of chronological relation; in order to create a continuous and non-repetitive narrative. The categories that test interactive narratives are robustness, controllability and the ability to keep the user engaged.

Creating an Intractable Narrative is the equivalent work for creating several non-interactive works, due to the large domain size required from which different narratives are born; which is a burdensome overhead. Automating the process of story generation is commonly used in video game industries, since most domains are limited in size. The procedural algorithms are not only used in narrative creation but can even be used in conjunction with other aspects of the game. Some examples of procedural generation used in games are: procedurally generated worlds (Minecraft, Elite Dangerous), procedurally generated world items (Borderlands), procedural generated audio ques (Half-life), procedural generated characters (No Mans Sky). The provided benefit of automation processes is to make each player's experience of the same game or narrative differently and uniquely. Developing a domain where different stories can be generated from is costly in terms of designer and developer time. However once the framework is established, the AI architecture can generate exponentially many content then the hand-crafted approach.

Critically acclaimed game Skyrim (Bethesda Softworks, 2011) has a hand written narrative for its main story. However optional missions are generated through procedural generation techniques, these missions are called Radiant Missions. The world can generate infinite amounts of these Radiant Missions. Radiant Missions are generated by the request of the player, and the game chooses a location and a target for the mission and alters the game world; creating a new mission and creating the illusion as if the mission was already in the game. Resulting mission can be in any number of locations in the game world and any number of characters can be a part of the mission. Radiant Missions generate a unique mission for the player to partake in. By embarking the mission's journey, the player may encounter new areas or characters by simply trying to reach to the objective. These missions being infinitely generateable, Radiant Missions encourages the player to explore, thus enriching the gameplay experience. It extends the lifetime of the product through content generation.

While being a solid system, Radiant Mission system lacks in few aspects. In Skyrim, the main plot is hand crafted and Radiant Missions are a side addition. To be able to coexist with the already existing hand crafted main plot, Radiant Missions cannot affect the main story line in any way or form. If an assassination mission is generated through a Radiant Mission; its target cannot be any of the key characters from the main plot. Their demise would break the main plot, since the main plot isn't designed to handle scenarios

where any of the essential characters can have an untimely death. Therefore the game simply forbids such key characters and items to be targeted by the Radiant Mission system. Another lacking point of Radiant Mission system is its repetitive nature. Two missions generated by Radiant Mission system only differs by the location and the target. Because of these lacking points of Radiant Missions, players figured out the artificiality of the system, and the player feedback on Radiant Missions were negative; otherwise a very critically acclaimed game title. The negative reception would be opposite if Radiant Missions had some effect on the main story, thus players would take feedback from the game world as if the time and effort put into accomplishing the Radiant Mission would have repercussions on the world, assuming the player could kill a key figure in the narrative and change the whole plot of the story.

Dwarf Fortress is a prime example of how narrative planning can make a successful product. Unlike modern games where graphical fidelity is achieved using complex 3D models, animation capture software and high fidelity sound effects to captivate the audience; Dwarf Fortress has none of this, and is one of the games featured on Museum of Modern Arts in 2012. Dwarf Fortress being an ASCII game only communicates with its user by ASCII characters. It shows a bird's eye view to the land and its inhabitants only using ASCII and colors, where different ASCII characters such as may represent a dwarf and the character may represent a goblin. From an interview with its creator studio Bay 12 Games, "For instance, when you travel to certain cities in the game and speak to a merchant they might tell you that their leather caps are made in an elvish city half a world away. And it will be true. They really were made there, during world creation, and traveled to this market for you to buy before you even started playing."[1].

The captivating aspect of the Dwarf Fortress comes purely from the narrative experience it offers. Dwarf Fortress is a game which is built on top of an astounding world generation algorithm. Before the player starts the game, a world must be generated. This world creation process generates entire continents, mountains, caves, wildlife and mineral deposits. Game simulates geological events and records these generated data to be used later in the game and generation process. After world generation, narrative planning takes place where the game places civilizations on the created world; humans, elves, goblins, dwarfs. These factions wage war with each other prosper and fall, kingdoms are formed and trade routes are established, heroes with

legendary deeds are generated and betrayals are made, natural disasters occur. The whole generation process simulates thousands of years of time in the generated game world. All of this history is stored in the world for the player to discover. Only after the world generation is complete the player can embark upon a new journey in this rich environment ready to be explored. In Dwarf fortress player takes control of a dwarf colony. Where player must make decisions to expand, secure and prosper the colony. Other factions may decide to wage war on player or make a trade deal, all dependent on the actions of the player and the events that take place in the simulated game world.

Figure 1: Dwarf Fortress

Since Dwarf Fortress is a game of a constant struggle for survival, it lacks an end to its story. The ending for Dwarf Fortress is either the death of the dwarf colony or the player choses to stop playing. It's an unending test of endurance, where all world events are generated without a final goal in mind. Since the playtime of the game is significantly less than the simulation time of the world generation algorithm, the interactions of the user does not affects the narrative in an interesting way.

In our thesis we are proposing a hybrid AI architecture, Goal-oriented Hierarchical Task Networks (GHTN), combining different approaches of the two most popular AI architectures in gaming industry; Behavior Trees and Goal Oriented Action Planning. GHTN is a Hierarchical Task Network (HTN) based architecture, where Behavior Tree-

like approach to scripting and Goal Oriented Action Planning-like planning mechanisms is combined in harmony.

With GHTN we designed a case study for Interactive Narrative to explore the capabilities of the architecture. GHTN is also applicable to other domains of such as procedural generation and behavior planning. Interactive Narrative is one of the more challenging domains for study since it fully utilizes both the iterative and planning features.

1.1. Thesis Structure

The rest of this thesis is organized as follows.

Chapter 2 provides an introduction to Interactive Narrative as well as requirements from a good Interactive Narrative system. Chapter 2 continues on with AI architectures Behavior Tree, GOAP and Hierarchical Task Networks, their advantages and disadvantages.

Chapter 3 describes the modifications proposed to HTN, details the designing process for behavioral tasks, and discusses how Interactive Narrative can be applied to the proposed system.

Chapter 3.4 goes over the pre-planning and planning algorithms by simulating the algorithm on a predesigned task space, visualizing interactions and inner workings of the algorithm’s planning system.

CHAPTER 2

Background Information

2.1. Interactive Narrative

It has been witnessed during the recent years that there has been an increase in the development of training systems that are simulation based and has the capacity to engage multiple spectrums under it in order to cater the needs of the market (Magerko, Stensrud and Holt). For example in order for a pilot to learn properly how to fly an aircraft, simulation based training system would allow the pilot to learn flying an aircraft without having the need to practice over a real aeroplane. The simulation based training system would act like a real world aircraft which would help the pilots to enhance to learn or even to enhance their flying skills. These kinds of simulation systems has already been introduced in the market and has been catering different kinds of industries such as health care, business management, education, military etc. “human in the loop” simulation system is another name for simulation based training where synthetic environment is created for trainee in order for them to acquire the necessary skills and education through the use of the simulation system. Traditionally, the way of training people was very different as compared to the ways of current era. In the past there were no training systems as such or even if they were in place, they were not comprehensive enough to teach a trainee the necessary skills. Therefore trainees were provided real world scenarios and real world application to test their skills and increase their knowledge which was also very expensive (Hill, Gratch and Marsella) (Faria, Hutchinson and Wellington). Compared to the current situation, such costs can be avoided through the use of comprehensive simulation systems which would allow trainees to gain necessary skills and knowledge affordably and in the least expensive ways. Such a simulation systems would also allow interactive virtual experience which not only enhances the skills of the trainee but also gives the trainee room for committing

errors which would not be possible when being exposed to a physical environment where the margin for errors is next to none. Hence better learning opportunities are available for the trainees leading to developing better skills and performance. Although the simulation systems have gained increasing popularity over the years yet there are a lot of challenges being faced for developing a comprehensible simulation based systems. Training refers to exposing a person to different number of scenarios or sequence of events where a trainee could enhance their multiple skills. This is one of the most critical elements of a simulation system since through the use of it, multiple objectives of training are achieved. Being able to be exposed to different scenarios, trainee is able to undertake multiple training sessions and certain training missions to be able to enhance their skills effectively. However ensuring that the trainee is able to achieve the desired objectives, it takes a lot of time since manual authoring of multiple scenarios is one of the bottlenecks being faced during the training sessions. Moreover, care has to be taken about how the scenarios will be executed while ensuring the actions taken by the trainee influences the outcomes of the scenarios and helps to progress the training accordingly (Zee, Holkenborg and Robinson) (Riedl, Stern and Dini).

The other name for Interactive narrative is known to be interactive story telling which has now gained certain grounds as digital entertainment around the globe. Training domains are very actively taking interactive narrative into consideration where trainee or a player has the choice to unfold the story of the scenarios according to the actions which they take in the virtual world. The virtual environments now created are highly immersed which is also one of the visions of the interactive storytelling and allows creating dramatic experiences for the trainee by allowing them to influence and unfold the story according to their actions. Such an experience is also termed as Holodeck experience. Certain automated means like AI planning are also employed by most interactive narrative systems in order for them to generating narratives due to which the burden of authoring is alleviated. Although multiple areas of interactive storytelling are in their infancy and a lot of research is being conducted in this regard as well in order to improvise and improve the interactive narrative systems (Kato). Over the last twenty (20) years multiple interactive narrative systems are being developed and multiple techniques has also been offered in this regard over the years (Faria, Hutchinson and Wellington). The foremost challenge which has been observed during these years has been about balancing the need to coherently progress the story with the user agency.

Since progression of a story in multiple directions due to influential actions taken by a trainee requires deep understanding of what actions might certainly be taken by the trainee hence designing the outcomes for those scenarios requires comprehensive understanding to unfold the story and reach a conclusion. There are no best ways of knowing the intensions of a user since user has the options to act in a way which they feel would be best for them. Designing scenarios accordingly is one of the most challenging aspects of interactive narrative systems or interactive storytelling. The users are not determined to take predefined steps to unfold a story but they are more prone towards testing the limits of the story telling to learn how vividly a story might unfold and in which directions therefore such an uncertain situation creates challenges. These challenges are to be catered by balancing the competing needs of the individuals and allowing them to feel that they have the control over unfolding the story in the most appropriate manner while ensuring that the coherency of their experiences are maintained (Hill, Gratch and Marsella) (Riedl, Stern and Dini).

One of the solutions to cater to these challenges has been through the use of drama manager who ensures that the narrative is being driven forward according to certain models in place which ensures quality and experience for the player. The drama manager is also known as the experience manager who plays a vital role in influencing the actions of the character which are being controlled by the user. The drama manager ensures by way of intervening that the actions undertaken by the user are being implemented in the narrative. The drama manager actually interprets the future actions of the user controlled activities by way of future projections. The projections are not made randomly but through narratological principles and other criteria's through which the quality of the user experiences are ensured (Zee, Holkenborg and Robinson).

The fictional world can be in different states due to the actions being performed by the NPC's, users and drama manager. Considering the fact that NPC's had not been discussed previously therefore it would be feasible to understand the concept behind what NPC's are. Basically NPC's can be defined as the transitioning of the virtual world through the transitioning of the virtual world in different states by means of the actions of the NPC's just the way a virtual world would transition by the actions taken by the user. NPC's are more basically helping the user to implement their actions in the virtual world and drama manager is there to ensure that appropriate outcomes and experience is being generated by such actions being undertaken. All these criteria and agents are

being built by the human author in controlling the virtual since it is necessary to shape the experience of the user and since human author will not be present their to ensure such quality of experience therefore such agents are in place to implement the actions and shape the user experience accordingly. Considering the importance of the human author and drama manager, it is necessary that a relationship exists between them in order to ensure that the concerns of the interactive narrative research are being catered vividly (Riedl, Stern and Dini). The whole scenario can be explained through the figure 1 which shows various transitioning states and outcomes of multiple actions being taken by the user to unfold the story while ensuring quality of their experience.

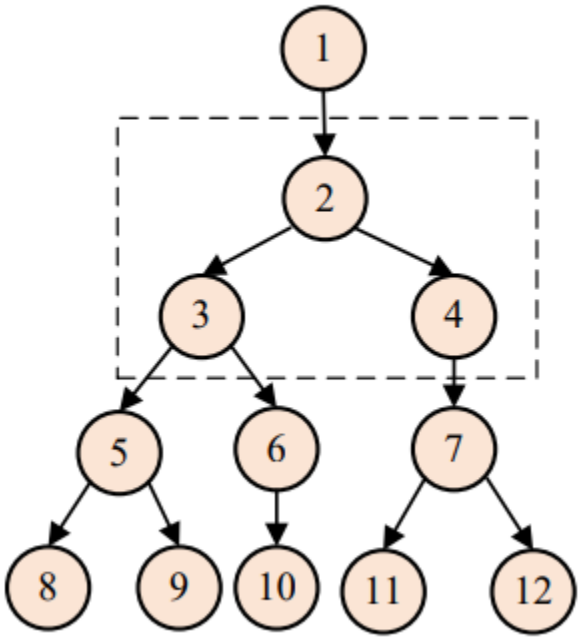

Figure 2: The unfolding of the story through tree graph representation

The figure shows different outcomes of a single story due to multiple actions being taken by different users. Since these outcomes vary widely due to having users to pursue different kinds of tactics to explore the dynamic nature of the interactive narrative systems therefore multiple scenarios are being created to overcome that challenge. Also in order to ensure the quality of the user experience, drama managers are built in place to ensure that the outcome of an action doesn't skips a path and lead to a farther outcome for an action hence drama manager ensures that the story unfolds in a logical manner and not jump from path 3 to path 8 directly or from path 3 jump to path

7. If such drama manager were not being put in place then the quality of the user experience would have been compromised since the outcome of an action to pursue path 5 would be against it and path 7 would be pursued. In such a situation drama manager ensurs that the outcome of path 5 should be path 8 or path 9 only. Also it is not necessary that the story unfolding from path 5 will have an entirely different outcome and different story if being pursued. Different paths could lead to a same story as well which can be explained through figure 2 as follows

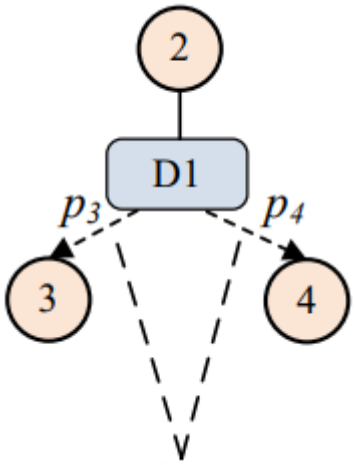

Figure 3: different paths leading to a same outcome at a later stage

Figure 2 explains the fact that the conclusion of different paths could be the same but in certain situations they could be different as well. Hence it is not obvious that different paths will have entirely a different story but the story could unfold in entirely a different way while leading to a similar or same conclusion at the end. All these paths and trajectories are being designed by the human author but as drama manager acts as a replacement for the human author in interactive narrative systems.

2.1.1. Key requirements of Interactive Narrative

For the purpose of the development of the interactive narrative system, there are certain key considerations that need to be followed to ensure quality experience of the user. Two categories have been found for the purpose of key requirements in developing interactive narrative systems which are as follows:

The first category deals with the perspective of the trainers about how they would want an interactive narrative system to help them achieve their desired goals. In this regards robustness and controllability of the system are being tested.

The second category deals with the perspective of the trainee in which the trainee considers how much engaging the experiences could be through the use of interactive narrative system. In this regard personalization and interaction is being taken under consideration.

Under the first category, controllability is focused towards how the desired outcomes are being achieved through the use of the interactive narrative system to meet certain training goals. Whereas robustness is considered as the robustness of the outcomes in a virtual world. Since it is a known fact that there are multiple actions that could possibly be performed by the user which has different outcomes but just to explore that possibility of achieving exceptional outcomes, users may perform certain actions which could lead to outcomes not being considered by the interactive narrative system and has might lead to undesired outcomes. Such undesired outcomes are to be avoided at all costs and such considerations are called robustness.

Under the second category personalization refers to outcomes that are being preferred by the individual users or trainers according to their needs whereas interaction refers to influencing the storyline of the interactive narrative system for unfolding the story (Kato) (Riedl, Stern and Dini) (Zee, Holkenborg and Robinson).

2.2. AI Architectures

AI research starts with asking a question to figure out which AI architecture is the most feasible solution to the problem at hand. In the process of considering different architectures, Behavior Trees are the first architecture to consider. The following section explores Behavior Trees and other the most frequently used AI architectures.

2.2.1. Behavior Trees

Behavior Tree (BT) is an AI architecture, which is used to implement complex sequences of events. BTs consist of two parts; the BlackBoard and the tree.

The BlackBoard is a globally accessible bundle of states, which represents the current state of the world. The nodes in the tree updates BlackBoard globally, the BlackBoard never has a local copy anywhere.

The tree is a branching list of nodes, originating from root to leaves, where branching appears in the presence of multiple paths from a single node. The tree is not balanced and number of children varies from node to node. In the most common implementation;

- The root node is insignificant, is mostly used as a pointer to the tree
- The internal nodes are a bundle of expected BlackBoard conditions. They may contain other internal nodes or leaf nodes as their children.
- The leaf nodes represent actions, which can be taken.

BT is a reactive system, which takes the BlackBoard as a parameter, and iteratively works from the root to the leaves. The iterations work very similarly to depth first search (DFS). At the start of every depth level, every children of the current node is evaluated left to right until one with valid preconditions (with respect to BlackBoard) is found. When the preconditions match, the depth is increased and search is done for the matching node. When no preconditions match on that level, the recursion returns false and search is continued on the parent node again. In the most common implementation, the search stops when a leaf node is executed. In more advanced implementations, there

are modifier nodes (selector and sequence) so that the search may last until a leaf is executed, or a sequence of leaves is executed.

Behavior Trees have no Goals, they work once or they are run multiple times until an external stop command is given.

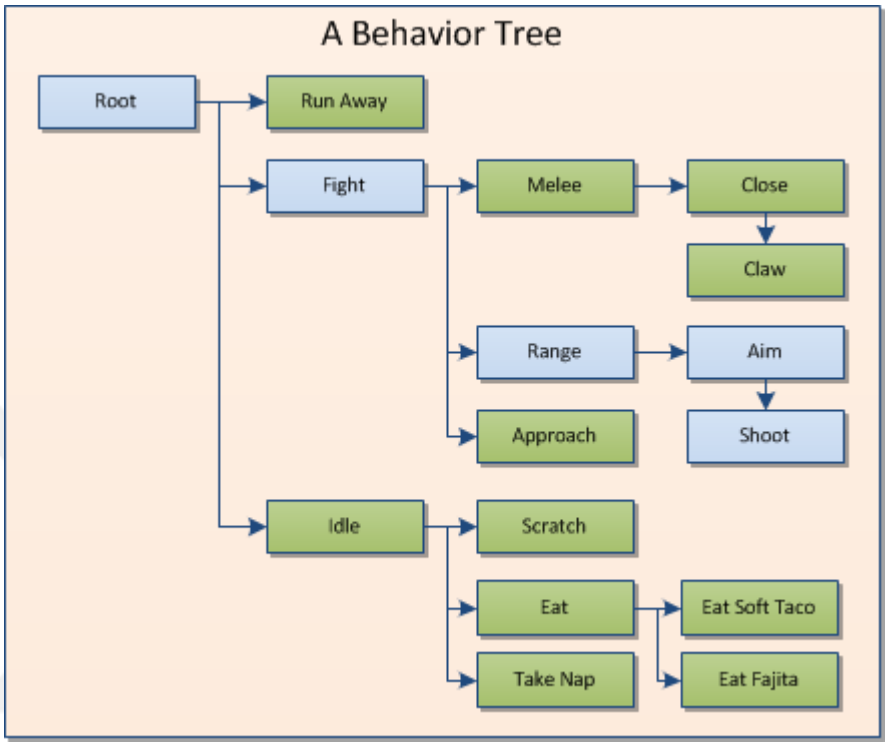

Figure 4: A Generic Behavior Tree

Advantages of BTs

BTs are often the most suitable architecture to solve the problem in hand due to the fact that they are simple to code, easy to design and somewhat scalable.

It is relatively easy to prototype a design in BTs due to its simplicity and straightforward nature. Behavior trees also have many different implementations being available widely over the game engines, most of which support GUI and drag and drop features, which enable non-coding background designers to design a system with ease. BTs are also the go-to AI architecture in most computer games due to the nature of game AI having a low count of behaviors and cause and effect relations being simple.

Disadvantages of BTs

BTs can scale from tens of nodes to few hundreds of nodes; however it is very infeasible to develop a behavior tree further in the node count. In an oversaturated Behavior Tree, the tree itself becomes unmaintainable since it becomes harder and harder to read, understand and create relations in. When a new node is designed to be added, its preconditions should be decided and the newly designed node should be attached to another node in the behavior tree. The attaching operation is complex, since the whole tree should be considered while adding; the node may require to be added multiple times to different parts of the tree. While attaching to its parent, the location of the new node with respect to its sister nodes is also important due to the fact that the nodes are evaluated left to right.

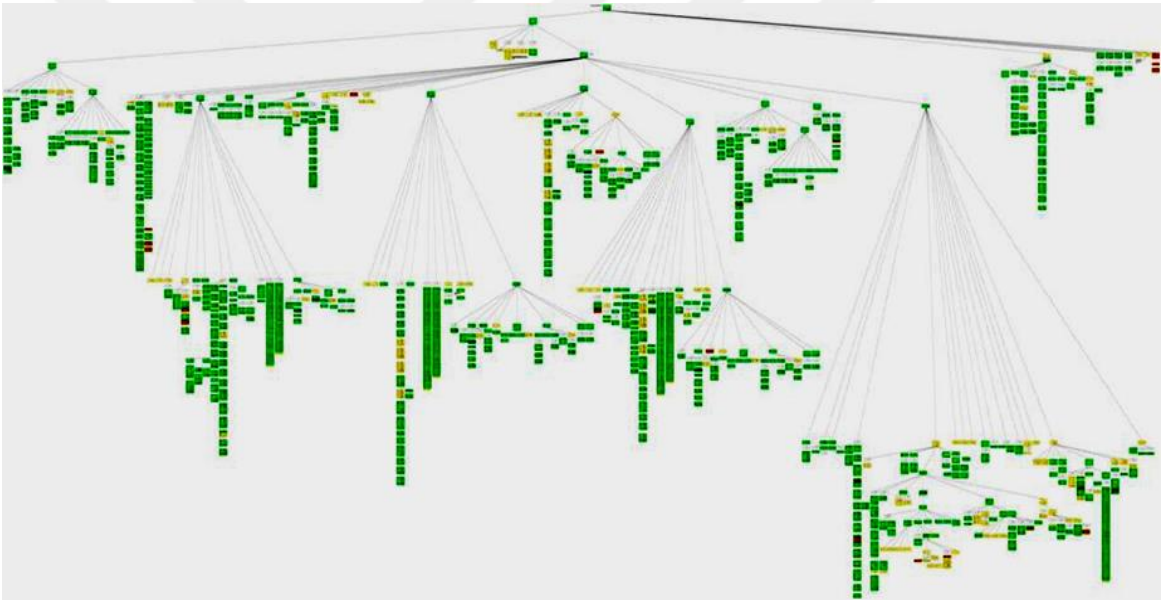

Figure 5: Oversaturated Behavior Tree with a Thousand Tasks

Since the BT algorithm works iteratively; the algorithm cannot undo operations and go back to previous world states. Imagine the ruleset of Towers of Hanoi; this task is unsolvable in BTs unless you give the mathematical solution in the format of a tree. If a new disk is added to the ruleset, the whole mathematical solution must be updated in order to meet with the new requirements. BTs cannot solve such a problem without the full mathematical solution since they are not capable of planning. BTs are best used in iterative problems where it can act reactively to the BlackBoard.

According to Damian Isla, Lead AI Developer at Bungie, “Hackability is key” when dealing with BTs. In his proceeding in Game Developers Conference, a very prestigious conference for its domain; Isla explains different approaches to BTs and how to modify BT’s flow with modifiers he calls “Stimulus Behavior” and “Behavior Impulses”. These implementations create callbacks within the BT and force it to handle certain cases. Whilst his propositions are perfectly valid and solve main problems of BTs, they do not contribute to scalability factor, creating what he calls the “Parameter Creep”; rendering the maintenance of the BT tougher over time.

2.2.2. GOAP

Goal Oriented Action Planning is proposed by Jeff Orkin in 2006, and was used widely on many classic computer games until 2012. Orkin’s research was phenomenal in its time, taking the focus off of script-like architectures (BTs), and putting it into planners (GOAP) in Game AI development. Orkin states that “The planning system that we implemented for F.E.A.R. most closely resembles the STRIPS planning system from academia.” Orkin states 4 main differences between the algorithms, however we will not explicitly cover these due to our scope.

GOAP consists of three parts; the world state, actions and a planner.

World State is a bundle of state variables bundled together. Initial World State is the world state at the beginning of the algorithm, and Goal World State is the expected world state at the end of the algorithm. It serves the same purpose as BlackBoard serves to Behavior Trees, however while a BT has one and only one BlackBoard, GOAP can have multiple World States.

Actions are nodes available in the planning space. Each action has a precondition, an effect, and cost. In order for an action to execute, its preconditions must be satisfied. When the preconditions are satisfied, the world state is locally updated. The cost of an action is higher for difficult tasks, and it is an arbitrary number greater than or equal to 0. GOAP algorithm does not have any physical structure such as trees; there are no connections between nodes.

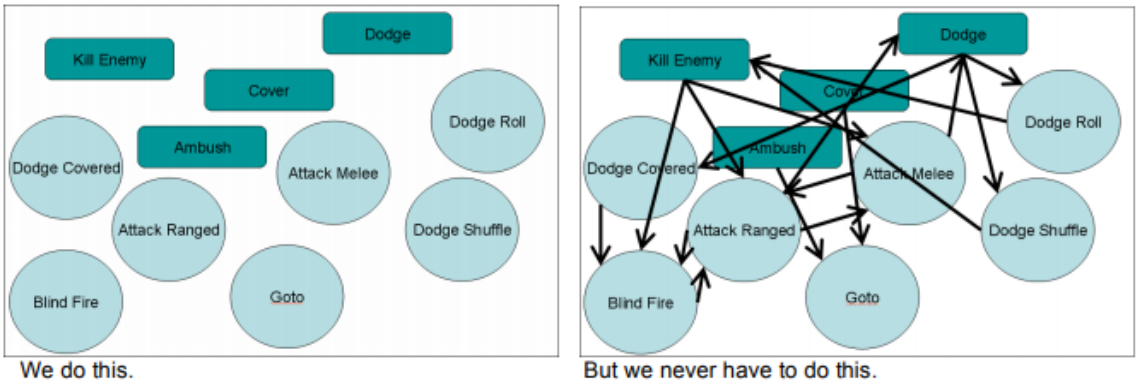

Figure 6: GOAP explanation from its creator, Orkin

The planner in GOAP uses A* search algorithm, to find a "path" between the Initial World State and the Goal World State. The A* algorithm utilizes the costs of the actions as a heuristic, and at every expansion utilizes the lowest cost action. In order to find a path, the algorithm starts from the Goal World State, and backtracks into the Initial World State. The path is the ordering of actions, there can be a path between two actions if the precondition of the one action is satisfied after the action is executed on World State.

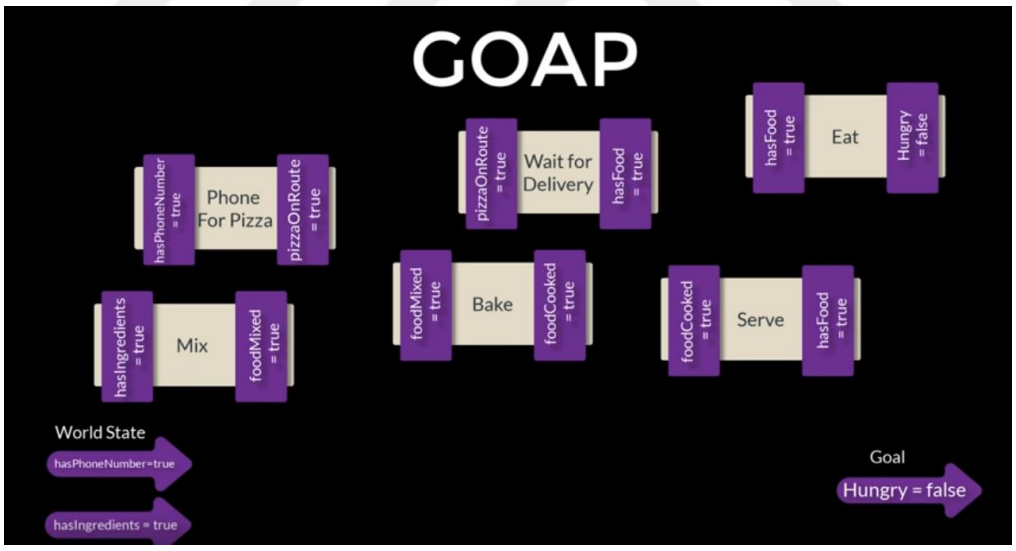

Figure 7: Planning to Eat with GOAP, Domain Figure

Since the algorithm works in reverse, planning starts from the Goal World State, and picks the first action which creates this state. Runtime of the example above is a following:

World State:

- hasPhoneNumber = true
- hasIngredients = true

Goal:

- Hungry = false

The algorithm starts with the Goal, and finds a path backwards to the initial world state.

1. The action “Eat” is chosen and added to the plan because its post conditions (Hungry = false) is satisfied.
    a. Local Goal State is updated to (Hungry = true, hasFood = true)
    b. Plan is [Eat]
    c. Algorithm continues, The Local Goal cannot be satisfied by some subset of Initial World State
2. The action “Serve” is chosen and added to the plan. In parallel, “Wait for Delivery” can also be added to the plan instead, we are not exploring that path for the sake of simplicity. Assume that the “Wait for Delivery” task is a high weight task so it is ignored.
    a. Local Goal State is updated to (Hungry = true, hasFood = true, foodCooked = true)
    b. Plan is [Eat, Serve]
    c. Algorithm continues, The Local Goal cannot be satisfied by some subset of Initial World State
3. The action “Bake” is chosen and added to the plan.
    a. Local Goal State is updated to (Hungry = true, hasFood = true, foodCooked = true, foodMixed = true)
    b. Plan is [Eat, Serve, Bake]
    c. Algorithm continues, The Local Goal cannot be satisfied by some subset of Initial World State
4. The action “Mix” is chosen and added to the plan.
    a. Local Goal State is updated to (Hungry = true, hasFood = true, foodCooked = true, foodMixed = true, hasIngredients = true)
    b. Plan is [Eat, Serve, Bake, Mix]

c. Algorithm continues, The Local Goal cannot be satisfied by some subset of Initial World State

5. The Local Goal can now be satisfied by some subset of Initial World State, the planning algorithm stops.

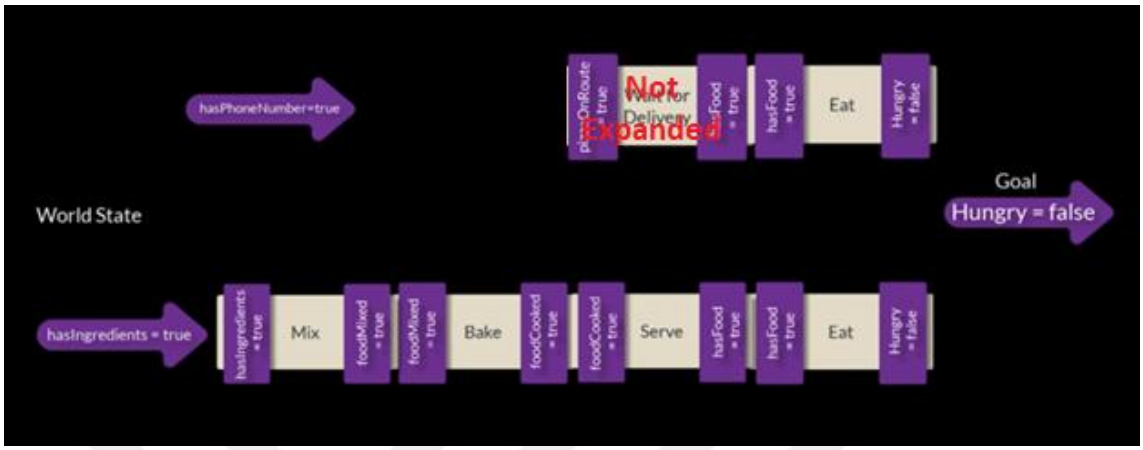

Figure 8: Planning to Eat with GOAP, Expansion Figure

At the end of execution, the algorithm returns the shortest path to the Goal from the World State.

Advantages of GOAP

It is an effortless task to add a new action to the planning domain. The action's preconditions and its effect should be decided, which is a trivial task; considering the fact that each action can be initially designed independent from each other. Determining the cost of the action is experimental since the cost should be in line with other tasks' costs, and the cost is the only parameter which determines the likelihood of the action being planned. An action with a very high cost would be chosen rarely by the planner, while an action with a low cost would be picked more often; since it does not dramatically worsen the heuristic. Related to this topic, Orkin gives the following example: Consider a new action TurnOnLights with the effect LightsOn=true is designed. If the TurnOnLights action is to be added to our planning domain, all that is required is to add LightsOn==true as a precondition to another action MoveAround. Therefore, actor will make sure to call TurnOnLights before calling MoveAround, to be able to navigate when they are in a dark environment. For every Goal which requires MoveAround action, it can be guaranteed that TurnOnLights action is called; meaning the lights are turned on, if they were not already on.

Disadvantages of GOAP

In GOAP, when an action is selected, it is removed from the list of available actions to prevent the repetition problem, however this is a two edged sword. Simple different tasks would require implementation differences to be solvable by the algorithm. Consider the case where the AI is required to collect 3 indifferent stones, and there is an PickOneStone action the available actions. If the planner picks the action PickOneStone, it will be removed from the available actions, and the plan would be unsolvable, even though there is a simple solution through repetition. Another approach would be implementing PickFirstStone, PickSecondStone and PickThirdStone actions; however such approach is not scalable.

In GOAP, creating new actions and adding them to the planning space is an effortless task. The absence of a real structure, such as not being in the form of a tree or a state machine, causes complexities if the designer wants to give some input to the planning process. Since there is a lack of higher structure, it is not possible to intervene and dynamically modify the search space. The designer can always edit weights for the heuristic dynamically; however this is not a trivial task since all the actions are only weighted by that one single metric. Such approach is very problematic since the planning ambiguity of the system makes it very fragile, modifying a weight value of an action will affect many actions that rely on the modified action, without the intention. GOAP is a black box in this sense, and tinkering with the ambiguity will cause more harm than good in a domain with high number of tasks.

2.2.3. Hierarchical Task Networks (HTN)

Hierarchical Task Networks (HTN) is a planning based, Goal oriented AI architecture. While having limited applications in the Game AI research, HTN is an architecture which is a good mix of the two most popular algorithms, BT and GOAP. Structurally, HTN is composed of multiple trees with height 1; and the planner algorithm jumps from tree to tree to find a solution, depending on the ongoing internal state. In video game industry HTN is used in many franchises such as Killzone, Max Payne, Total War and Dark Souls.

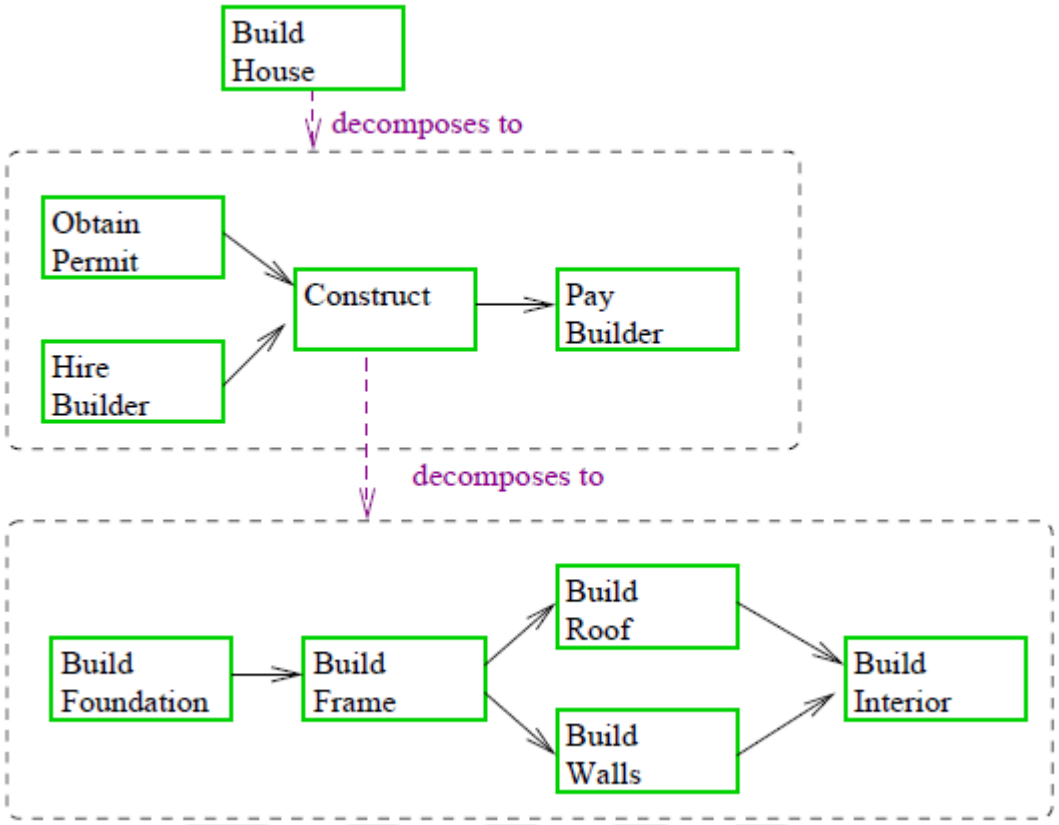

Figure 9: HTN Visualization of Tasks

HTN consists of multiple parts; World State, Primitive Tasks, Operators, Compound Tasks, Methods and the Planner.

World state is a bunch of states bundled together, which represents the current state of the world. In HTN, there is a single World State throughout the entire execution. The World State is changed by Operators.

Primitive Tasks are basic tasks which includes one or more Operators. All the Operators in a Primitive Task are called sequentially.

Compound Tasks consists of Methods, which are possible ways for solving that Compound Task. A Compound Task can have one or multiple Methods. When a Compound Task is executed, the algorithm tries to find a Method where its conditions are satisfied. Similar to BTs, HTN algorithm starts from the leftmost child Method and moves towards the rightmost Method until it can successfully run a Method. If there are no methods in a Compound Task that are satisfiable, the Compound Task does nothing. Compound Tasks are 1 height trees. The root node has the name of the Compound Task, and each of the leaf nodes correspond to a method.

The planner is given an Initial World State, a Task Array to start running from and an optional Goal State. The planner stops execution when the Task Array at hand is emptied; meaning that all the tasks that this planning operation was responsible for, are completed. If a Goal State is given, the planner will stop if its World State reaches to the Goal State. It is also possible to limit the planner by iteration count. At the end of execution, the planner returns the list of Primitive Tasks(These include Operators) so the World State changes can be applied one by one.

If the Compound Task "Eat" has two methods, CallPizzaDelivery and BakeCake, the leftmost child of the parent will have the first opportunity to work, in this case CallPizzaDelivery's preconditions will be checked first. If the preconditions do not match, BakeCake's preconditions will be tested. If the preconditions do not match for BakeCake as well, nothing will be done in this method. However if one of these Methods execute, they will add new tasks to the Task Array.

Advantages of HTN

The HTN algorithm is extendable due to the constant tree depth of 1 and extensive decoupling of tasks, methods and operators. Being able to group Methods under Compound Tasks allows the designer to create an internal order of execution mechanism inside Compound Tasks. Also, Compound Task, Primitive Task and Operator decoupling from each other allows the designer to freely implement any new task or operator without having to worry about previously implemented ones, and ease the complexity of debugging.

Hierarchical Task Networks are composed of 1 height trees; therefore they keep the initial ordering between nodes that is set by the designer. This allows the designer to create an order of importance to Methods sharing the same Compound Task before the planning phase, which will be kept and respected throughout the planning phase.

Expanding the search space in HTNs can be done through implementing a new Method to an already existing Compound Task, Creating a new Compound or Primitive Task or implementing a new operator. Adding newly implemented features require a single pass through the whole domain. Tasks are very easily debug-able since starting Task Array can be set to start with the newly implemented task, or the task that calls the new task.

Disadvantages of HTN

A planner exists in HTN; however the planner is not a heavy-duty planner as in GOAP. The HTN planner iterates over the network and checks conditions, accumulates the methods and updates a local copy of the initial world state until the local copy matches the Goal State. Wrong ordering of Tasks and Methods may cause the planner to never come up with a valid plan to reach the Goal State, even if logically the Goal State looks possible given the domain.

CHAPTER 3

Methods

The proposed algorithm Goal Oriented Hierarchical Task Network (GHTN) is a hybrid architecture for AI programming, utilizing both BT-like and GOAP-like approaches together. Unlike BTs, GHTN algorithm is fully fledged in capabilities. The unmodified BT algorithm is not plug-n-play level usable, and most real world implementations (such as Unreal Engine 4) implement parallel tasks, callbacks and other functions to extend the system for general use. The GHTN algorithm is designed to be capable of any foreseeable technical requirements in the domain of game AI, without any need of extensions.

This thesis focuses on simplifying the HTN algorithm, extending its features with extensions, and implementing a secondary GOAP-like planner for path finding in the behavior space.

We will be using the domain “Interactive Narrative” to explain the GHTN algorithm. The GHTN algorithm provides a discontinued, non-repetitive and interactive framework for the chosen domain.

In methods, we will explore the modifications on the HTN architecture and implementation details, with respect to the requirements of our domain of application, Interactive Narrative. The format of the Methods section will be as following;

- “Section 3.1: Goal Oriented Hierarchical Task Networks” explains the algorithm of GHTN architecture without any domain specific explanations.
- “Section 3.2: Designing Task Network” explains how to design a task network to better utilize the features of GHTN algorithm.
- “Section 3.3: Use of GHTN in Interactive Narrative” application of GHTN to Interactive Narrative is discussed and domain specific examples are explored.

- “Section 3.4: Further Research” discusses how the algorithm can be improved and what are similar methods used in planning and interactive narrative research.
- “Section 3.5:Full Algorithm Overview in a Test Environment” demonstrates a use run case for both the planner and iterative parts of the algorithm.

3.1. Goal Oriented Hierarchical Task Networks (GHTN)

The HTN architecture explained in the background section of the thesis is the most common architecture found in most HTN implementations. GHTN takes the HTN algorithm as its base algorithm; however we have gone through several structural modifications to HTN while minimizing the impact on the flow of the algorithm. The reasoning behind the proposed modifications can be explained as the following;

- The AI algorithms explored throughout the thesis are mostly used in game AI; they are designed for the use of small and confined behavior counts. The nature of our work requires more extensive amounts of behaviors in order to be able to create different plans within the same behavior space. Simplifying the already existing HTN terminology will allow us to add our terminology for new terms without minimizing any complications.
- The HTN algorithm is very modular but complex in structure. The structure can be simplified without altering the workflow, but hurting modularity. To increase modularity, free functions can always be used when designing the behavior space. In HTN, the excessive modularity increases the boilerplate code and design required just to do very simple tasks; complicating the design process.

3.1.1. Base HTN Algorithm Simplifications

In the GHTN algorithm, Compound and Primitive Tasks are merged into Tasks. Like the HTN algorithm, a task can have one or multiple methods and tasks can optionally have a default method. To simplify the planning and designing flow, tasks can modify world states as if they are calling other tasks.

To better differentiate tasks that do not add any sequence and tasks that fail; tasks can now return an invalid state when they fail. A task without a default method will fail if none of its methods' preconditions' holds. The existence of invalid state relaxes the requirements of the task network designer, simplifying the flow from designer's creativity to algorithm's expected format.

3.1.2. Goal

A Goal is the expected end of the algorithm. The algorithm works until the Goal is satisfied, and stops on the instant the Goal is satisfied. Reaching the Goal is not the end of the algorithm since Goals can be given to the algorithm in sequence, one after another.

A Goal can be expressed to the algorithm as a bundle of states, bundle of tasks or categories of tasks. As long as it is possible to convert the input to a bundle of world states, anything can be expressed as a Goal. Combination of all the above mentioned Goals can also be combined. Even though all the above mentioned types of Goal input are supported, using tasks and categories to create world states require some knowledge of the designed world space. These options can cause instability in the task network by creating special cases in the preparation algorithm, which are to be explored in Section "3.1.4: Preparing Behavior Space for Goals".

Goal as a bundle of states can be represented as an array of world states. Goal States are expressed as [bCondition1 = true, bCondition2 = false]. The algorithm will attempt to find a sequence of events to satisfy both requirements in this state at the same time, or one after the other; depending on the behavior space created by the designers.

Goal can be tasks since tasks can be converted to world states. Goal as task is expressed as [NameOfTask1, NameOfTask2]. Task Goals are distinguished from state Goals since there are no "=" in the expression, and there exists one task with the name "NameOfTask1" and "NameOfTask2". The algorithm's first job is to convert the tasks into world states in order to express this Goal requirement with already existing systems. If the task has a single method, the preconditions of that method are registered as the Goal State. If the task has multiple methods, the preconditions are converted to Goal States, and they are registered as parallel Goals. These tasks are moved in front of other tasks in the behavioral space, so the Goal task has priority over every other task in the behavioral space. For every task in the Goal tasks, this conversion is repeated.

A category of tasks can also be a Goal State. In the behavior space, multiple tasks can be tagged with categories, and the planner can register that tag as a Goal. The Goal registration system for categories utilizes the same algorithm of task Goals; for every category in the Goal categories, the registration algorithm is run.

The algorithm of handling multiple Goals in parallel is explained in Section 3.1.10: Interactive Planning.

3.1.3. Tasks and Methods

While creating a behavior space, ordering of tasks is a vital topic which directly affects the outcome of system quality. The tasks are divided into two parts while designing; Root Tasks and Non-root Tasks.

Root Tasks are the tasks that directly attach to the behavior space root. These tasks can directly be called by the planner, can be used to initiate.

Non-root Tasks are the tasks that are not attached to the behavior space root; they are free task networks without a connection to the main behavior space. Non-root Tasks can be used in plans but cannot initiate plans, cannot be called by the planner directly but Root Tasks may bring Non-root Tasks as their subtasks.

3.1.4. Preparing Behavior Space for Goals

Preparation is done after every new goal input in order to maintain a healthy behavior space. Preparation trivializes many hard to accomplish problems, and eliminates the need for designer intervention in the runtime; thus making the algorithm self-sustainable. Through preparation, the system continuously reorders the sorting of tasks in behavior space, and creates the illusion of existence of a designer whom constantly mettles with the behavior space in order to create contextual questions. During the preparation phase, the algorithm marks the related tasks with the goal. The main aim of preparing the behavior space for new goal is to passively add direction to the behavior space by changing the ordering of tasks, thus the AI is more likely to perform the related tasks instead of non-related ones when multiple tasks' preconditions are satisfied.

Preparing the related tasks: The tasks which are related to the goal(s) are marked when a new goal(s) is registered to the system. Since GHTN is designed to handle

larger behavioral spaces, the number of possible actions which can successfully be taken is many. By finding related tasks directly or indirectly correlated with the Goal, we are moving them above every other task in the behavioral space in terms of importance for the story. Reordering of the behavior space allows the HTN-like left to right search to be used more extensively; therefore reducing the need for more complex search operations. The intractability system is greatly simplified by bringing Goal related tasks up front in the behavior space ordering.

The following algorithm is utilized for marking the related tasks. The approach is similar to creating a dependency graph. Starting with the Goal as states, the algorithm finds the tasks which modify those states. And from those tasks, the algorithm looks at their methods' conditions and finds the tasks which affect those states towards the goal. The algorithm stops when a new iteration starts with behavioral task count above %25 of total behavioral tasks. In other words, approximately %25-30 of the tasks that are directly or indirectly related with the states in the Goal are heightened in priority. The purpose of the percentage limit is to prevent extreme direction which would otherwise be gained through setting a Goal. Instead the algorithm stealthily grants direction through the existence of the Goal, which prevents bee lining to the Goal through the interactions.

In the existence of Goals as tasks or Goals as categories, the tasks are moved above every other task to priorities the taken actions when the any of the tasks' preconditions are met. This is a special case that breaks the rule for maintaining initial task order. Due to the fact that the rule is broken, using tasks and categories as Goals require some knowledge on the designed task network.

When the first Goal is completed and a new Goal is registered, the current ordering of the tasks is saved as the initial task order, and preparation for the new goal starts again. Updating the initial Goal ordering is important since when the second Goal is entered the player has already built up their character in the narrative. Starting the second Goal should maintain the history caused by the first Goal to strengthen the bond between the player and the narrative. This helps the algorithm fulfill the requirement of continuity between multiple Goals.

Disabling the contradicting methods: disables certain methods which prevent the goal to be reached. If all methods of a task are blocked, the task is blocked from the behavioral space.

The following two examples explain how a method is found contradicting;

For the both examples, assume the following behavior space, which reflects real life. The goal is to play the guitar in the end of year concert, and be hungry at the same time.

In the above scenario;

- Eat task makes the player not hungry, but there are tasks that make the user hungry again such as; sports, working and sleeping. The task eat is not contradicting even though it negatively effects one of the goal states. Since the effects of eat can be negated by different tasks in order to reach the goal, it is not considered as contradicting with the goal.
- The task car crash makes the player's wrist injured so they cannot play guitar or drums ever again. Since there are no tasks which reverses the wrist injury, the car crash task found contradicting and it is disabled.

In other words, a method is disabled when there is a Goal [stateX=true, …] and the initial world state is [stateX=true, …] and consider there are methods that can update stateX to false, but there are no tasks that update stateX to true. In this case, if stateX becomes false at some point in the AI execution, it can never go back to true. In such case any method that effect stateX negatively gets disabled in the behavior space. The disabled methods are restored when the current Goal is completed, and reapplied when a new Goal is registered.

3.1.5. Trigger

Triggers have multiple uses, one of which is to simulate external events on the system. Triggers are special tasks which have priority over any other task in the task network. In game AI, it is often that AI perception is decoupled from behavior AI to simplify the system. Triggers are GHTN's endpoint to satisfy such requirement, for example if an AI Perception module exists in the system; it can be linked with GHTN through Triggers.

Another use of Triggers is similar to the observer pattern in software engineering. A Trigger can be created on a state to catch the exact moment the state has changed. This capability enables the designer to create related world states, and the designer can create cause and effect relations on the world states.

Triggers are decoupled from every other task to simulate the dynamic structure of the world in the task network. They are the leftmost child of the root, coming before the behavior space. Their uses in design and narrative generation will be explained more in depth in "Section 3.2.3: Trigger Selection and Design" for designing task space for uses in interactive narrative.

```
GoalPreperation(Tasks, Initstate, Goalstate)
    Tinit ← Tasks
    Tasks ← DisableConflictingMethods(Tasks, InitState, GoalState)

    important ← []
    foreach curTask in Tasks
        foreach curMethod in curTask.methods
            if not curMethod.disabled and isRelated(curMethod, GoalState)
                Add(important, curTask)
                Remove(Tasks, curTask)

    TempGoal ← []
    while important < totalTaskCount * 0.25
        foreach curElevated in important
            foreach curMethod in curElevated.methods
                foreach curState in Satisfy(curMethod)
                    Add(TempGoal, curState)

        foreach curTask in Tasks
            foreach curMethod in curTask.methods
                If not curMethod.disabled and isRelated(curMethod, TempGoal)
                    Add(important, curTask)
                    Remove(Tasks, curTask)
    Tasks = important + Tasks


Satisfy(method)
    state ← []
    foreach curPreeonditions in method.preconditions
        Add(state, (curPreconditions. key + eurPreconditions.operator + curPreeonditions.value))
    return state

isRelated(method, coalstate)
    foreach curPreconditions in method.preconditions foreach curKey in curPreconditions.key
        foreach state in GoalState
            If state.key • curKey
                return true
    return false
```

Figure 10: Preparation Algorithm

3.1.6. Task Queries

When a task is queried, the system checks what action that task takes in the current world state, without executing the task itself. Since a task can return a valid plan, or return invalid; we can understand whether or not that task fulfills some custom requirements by the HTN system. Queries can also be created using custom local world states to deduct how the task would respond to such change in world state. Queries are mainly used by Trigger system and the Interactive Planning algorithm. For example, a Trigger can call a Query on a function to learn how that function behave with the upcoming change to the world state; while not disturbing the flow of the planner or the Interactive Planning algorithm.

3.1.7. Categorization

Tasks can be tagged with multiple categories, and the categories can be used as a Goal. Categories allow the possibility of having a broader Goal, therefore breaks the uniformity of always having a single Goal. Categories also create a more hack-able system thus improving the freedom of design. In the domain of game AI, thanks to categorization through tagging, the designer can easily implement “ThrowHandGranade” and “ThrowExplosiveBarrel” tasks with the tag “AreaOfEffect”. The planner understands that both these tasks have area effects, thus can create dynamic and different plans when an area effect attack is required. If the categorization system is not implemented, such tasks would still be created in the system using different world states however that would overcomplicate the design progress.

3.1.8. Preparing the Behavior Space for Planning

In the notation of HTN, the second method requires the first method to fail. The HTN tasks must be prepared for planning in order to fix such precedence problems which are only visible to the planner. It is similar to flattening in machine learning; the tree structure must be reduced into a one dimensional vector.

Consider the following task "GoCinema". This task has some romance and band related events. If the player satisfies any of these conditions, unfold happens and the end of the story is written;

```
Task(GoCinema)
    Method(inRelationShip = false, miaInvited = true)
        subtasks[miaRomantic]
    method(inRelationShip = false, sullyInvited = true)
        subtasks[sullyRomantic]
    method(bandInvited = true)
        subtasks[bandProblems = false]
```

The first method is always ignored since there are no methods above to compare it with.

The algorithm sequentially checks the methods from top to bottom, adding negation of previous methods' preconditions' to the lower priority ones

```
  Task(GoCinema)
  1 method(inRelationShip = false, miaInvited = true)
      subtasks[miaRomantic]
2.1 method(inRelationShip = false, sullyInvited = true, inRelationShip = true)
      subtasks[sullyRomantic]
2.2 method(inRelationShip = false, sullyInvited = true, miaInvited = false)
      subtasks[sullyRomantic]
3.1 method(bandInvited = true, inRelationShip = true, inRelationShip = true)
      subtasks[bandProblems = false]
3.2 method(bandInvited = true, inRelationShip = true, sullyInvited = true)
      subtasks[bandProblems = false]
3.3 method(bandInvited = true, miaInvited = true, inRelationShip = false)
      subtasks[bandProblems = false]
3.4 method(bandInvited = true, miaInvited = false, sullyInvited = false)
      subtasks[bandProblems = false]
```

In preparation, the system knows the exact conditions required by the flattened methods, and the precedence problems are solved. The following problem is avoided;

Consider GoCinema:Method3 is selected by the planner. The condition "bandInvited =true" would be added to the planning world state. Once planning is complete and the user is going through the plan, we can guarantee that the task "GoCinema" will be called, however we cannot guarantee that method 3 of "GoCinema" will be called. The world states may resolve the task by using method 1 and method 2. The reason of this complication is the following. When the algorithm was on the planner phase and the method is called with an update to planning world state, the system does not make sure that method 3 will be called without interference of the methods above itself. While planning for the conditions of method 3, the system must also make sure that methods 1

and 2 will not be called. By preparing the behavior space for the planner, we solve this problem by preventing higher in priority methods to interfere with the system by making programmatically adding new preconditions to force the methods to be mutually exclusive.

3.1.9. Planner's World States

The planner must maintain its world states while planning. In our domain, one task may change a condition, which was required by a previously already planned task. The planner has to understand this unwanted change, and add additional tasks to negate this change while planning. To accomplish this, the planner has to keep a log of the world state changes during planning. The list is kept ordered to allow the planner to understand cause and affect relationships between tasks. The world states that are not included in this list are not defaulted since they are unknown by the planner. Since this is a log, there can be repetitions of the same world state, even though the world states are conflicting. The planner maintains this list while planning, updating the world states with symbols to track progress. The planning is completed if every world states inside the list can be achieved. When the planner starts working, it copies the goal state into its local space to work on. Similarly to the GOAP algorithm(Section 2.2.2: GOAP), planning goes from the goal state to the initial world state. When the planner plans through some task's method, it updates the planner's world states to make sure that the method's preconditions will be satisfied at the point of execution.

Throughout the thesis, the "<=" symbol will be used to signify the current iteration in the planner's world states list.

The "." symbol signifies the state has been recently added to the list. The "." symbol is used solely for easier verbal explanation. Existence of "." and no symbol is identical for the algorithm.

There are 4 symbols available symbols;

- "!" Registered state - can be considered as a local variable which was introduced directly by the iterator "<="

- "%" Registered state through subtask. – can be considered as a local variable which was introduced indirectly by the iterator “<=”
- "+" Satisfied in the planner’s initial world state or with “!” or “%”
- "$" The other tasks of the subtask which includes the goal, these are ignored

More extensive example will be explored on “Section: 3.5: Full Algorithm Overview in a Test Environment”.

```
Planner world states:
+inRelationShip = false
.miaInvited = true  <=
miaImpressed = true
!miaGirlfriend = true
%inRelationShip = true
```

This list is taken from the middle of a planner execution. The currently explored state is miaInvited = true.

Consider that the A* algorithm proposes CoffeeShop:Method1 as a fitting expansion.

```
Task(CoffeeShop)
    method(attendedPlay = true)
        S-> "At the coffee shop Seb sees Mia's friend from
        the play. Friend tells Seb to maybe invite Mia to
        cinema. You do so. Mia accepts."
        subtasks(miaInvited = true)
    method(miaMet = false)
        Q-> "Prepare your car and leave for the Coffee Shop
        to have a chill day."
        S-> "While on your way to the Coffee Shop, someone
        cut to your lane. You almost had an accident. The
        driving lady shouted at you as if it was your fault"
        subtasks(metMia = true)
```

In order to satisfy the state miaInvited = true, the algorithm adds attendedPlay = true to its list. Therefore CoffeeShop:Method1’s preconditions will be satisfied.

Since attendedPlay = 0 is the precondition of the method, it is added before miaInvited = true. Also since miaInvited = true is registered now, it gets the symbol "!"

```
Planner world states:
+inRelationShip = false
!miaInvited = true  <=
miaImpressed = true
!miaGirlfriend = true
%inRelationShip = true
```

Now the iteration jumps to the first element in the list, which is "miaImpressed = true".

3.1.10. Interactive Planning (Interactive Narrative Only)

Interactive Planning is used to determine how questions are asked. The algorithm is expected to create a constant number (N) of questions at every iteration, to not alter the natural flow of the interaction. If the algorithm cannot come up with the required number of questions (N), less questions are presented. If only one question can be created by the algorithm, it is not presented as a question. The single question is invisibly chosen as an answer, the world state is updated accordingly and the algorithm continues with the preparation of the next question.

Depending on the existence of an active Goal State, the algorithm works differently in order to provide contextual questions.

When Goals are not yet activated or the last active Goal is completed; there are no active Goals in the system. In the absence of a Goal, the algorithm utilizes the HTN tree. Sequentially, Queries are sent to the children tasks of the main task node, with respect to left to right ordering. The valid plans that Queries return are collected and when the collected number of plans reaches N, these valid plans are asked to the user as if they are questions. Through the preparation algorithms, the questions are guaranteed to be related with the world context. Creating questions from tasks will be explored in “Section 3.3.6: Interactive Planning”.

In the event of an active Goal State, the Interactive Planning algorithm acts completely differently. The algorithm searches for N different valid sequence of tasks to the Goal State. These paths cannot be calculated with the basic planner of HTN, since the planner works sequentially and it does not have a search algorithm. The problem is very similar

to the problem that GOAP tries to tackle; therefore we use a similar approach to what GOAP algorithm does.

In this mode any HTN based logic is disabled until questions are formed, only the tree data structure is used. From the Goal State, the algorithm works backwards to the current world state using the A* search algorithm. A* is a heuristic based shortest path algorithm, mainly used in path finding in game AI. The application of a heuristic algorithm to a behavioral domain is done before by GOAP, and our algorithm is following the footsteps. From the Goal world state to the current world state, the A* algorithm is run. In the search, every children node of the main task node can be considered as a valid action. Only the children tasks of the main task nodes are considered because, if a node is not registered under the main task, it cannot be reached directly by the user. If this limitation is not given, the A* will create a plan which are valid, but the created plan will not be executable by the player. The heuristic value of A* is tied to the number of subtasks on the chosen task; the more subtasks a task has, the more questions are asked in the plan since number of subtasks increases the plan length.

Normally, the A* algorithm only return a single path from initial world state to Goal. We modified the A* algorithm in order to get N different plans to the same Goal. In interactivity, the user only sees the question which is formulated from the first or first few plan steps. Therefore after we get the initial path, we block the first task of the plan and restart the A* algorithm to get an alternative path. If the algorithm cannot find a path, this means that the blocked task had no alternatives; the first task is unblocked and the second task gets blocked and the A* algorithm tries again. We repeat this operation for N times, to end up with N paths. Formulating these plans into interactions is domain specific, which will be discussed in Sectio5: 3.3.8.

Since being domain and application specific, the algorithm to create questions from a sequence of plan will be explored at “Section 3.3.6: Interactive Planning”.

3.2. Designing Task Network Space

3.2.1. Ordering Tasks and Methods

The ordering of tasks should be made in a descending order, where the first task is the most significant task. In a single run only one task can work; the more significant a task is, more opportunity for that task to be used in iterative phase. If a task with easy to satisfy conditions are put into a high significance, it will be chosen very often. While ordering tasks, the following should be considered; “Given all of my tasks can work at the same time, can an order of significance be created”. The ordering of methods inside a task should be considered with the rules above.

While the iterative algorithm considers the task orderings; the planning algorithm does not consider the task ordering while planning; however it does consider method ordering.

3.2.2. Designing World States

World States are variables for the system to keep track of the current situation on the game world. A world state can have a boolean, integer or a string type. World States have default values; these values are enabled when an explicit value is not given to the state on the execution.

World states are written in the format of StateName=StateValue. For example; StateName1=true StateName2=13, StateName3=”Sebastian”. StateName1 is a boolean, StateName2 is an integer, StateName3 is a string.

Infinitely many world states can be created in the system however it is better to minimize the world state count to lessen the burden on the planner algorithm. There are mechanisms, such as Triggers, to simplify the work flow when there are too many world states for the designer to consider.

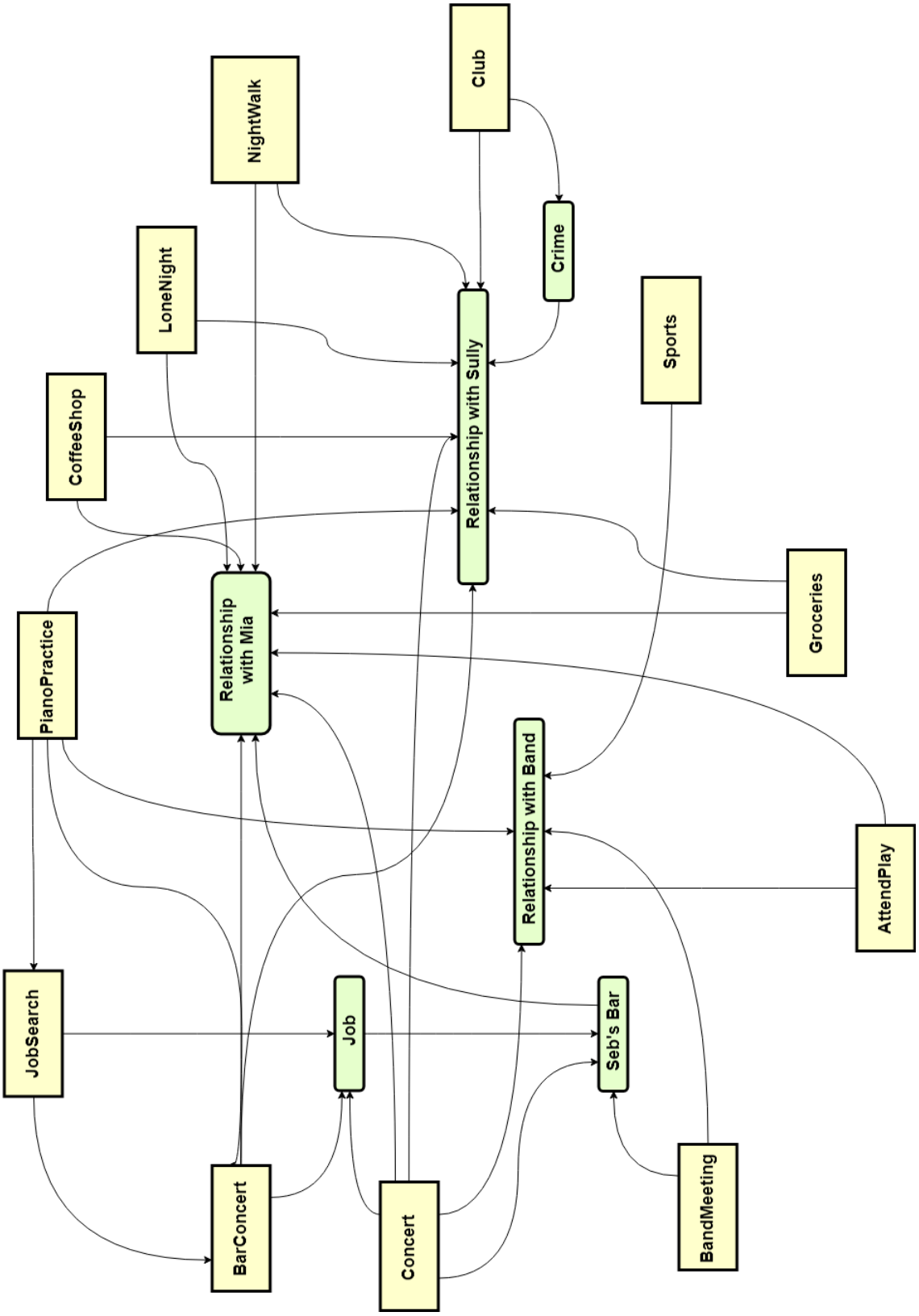

Figure 11: Relations of Tasks (yellow) with Green (world states).

3.2.3. Trigger Selection and Design

Triggers are useful when used with observer pattern of software engineering in mind. Consider the designers want to introduce lycanthropy to the behavioral domain, so characters can become werewolves. The preconditions of being a werewolf is [bFullMoon=”true”], this system is easily implementable through a simple task and without using Triggers. However the designer has to make sure that this task will somehow always stay higher in priority. If it is not always high in priority, reordering of tasks through preperation may cause other tasks to drop lycanthropy so behind in importance, there will be delays in becoming a werewolf or it might not even happen at all. Making this a Trigger instead will force lycanthropy on the instant that a change in bFullMoon is detected.

Another reason to make lycanthropy a Trigger is the following case; consider the behavior space is expanding and a new precondition is added to lycanthropy; characters can become werewolves if there is a full moon or they are angry. Now the behavior space has 2 states (bFullMoon and bAngry) linked to lycanthropy. At this point, it is more manageable to create a Trigger where if [bFullMoon=”true” or bAngry=”true”], bLycanthrophy becomes true. Consider 5 more possible preconditions are added to lycanthropy, instead of checking 7 world states before every lycanthropy related task, it is more feasible to just check for [bLycanthrophy=”true”].

3.2.4. Designing When To Use Queries

Through queries we can learn how a task would act in the current world state, decoupled from the planning algorithms. Consider the task “Concert”, the main focus of Sebastian’s jazz band. Through Triggers, the system can be designed to understand that concert is soon. However now the problem evolves into understands which characters can go attend the concert since if a member cannot attend, there must be a search for replacement or the concert will be cancelled. Running the task “Concert” will cause the agents to try to leave for concert, but it is impossible if there is a member missing in the band. We can solve this problem with using queries on the task “Concert” with a fake extension to the world state which says [AllAttending = true]. This will trick the

“Concert” task into ignoring the AllAttending condition; therefore through this query, the system can understand which band members are capable of joining. The query will return the subtasks of the satisfiable method, and the designer can deduct meaning from the subtasks. In the case of the example, the query will be sent once for every band member with the fake world state [AllAttending = true], the query will return the subtasks for those band member who can attend the concert and will return invalid for the band member who cannot attend the concert.

3.2.5. A* and The Integer Problems

Using integers as counters is problematic if a wrong approach is followed in the general design. Using integers as numbers does create several hurdles that we could not bypass. Mathematical operations (+,-,*,/) should not be made in subtasks of any task. Integers used as enumerations, or numerical integers that are not in any subtasks but are maintained through alternative solutions are also not problematic.

When the A* algorithm gets an integer state in the goal or while planning, to which there exist tasks that can do mathematical operations on them; the A* algorithm will endlessly try to use those tasks given the preconditions hold and the heuristic is low. This can be solved by using limiting operators such as >,<,>=,<= however the A* algorithm will still try apply the same method until the limiting condition return false. This is a massive exploration time lost for the algorithm; however the algorithm will still give meaningful results even if integers as numbers with mathematical operators are used. This problem will only hurt the run time and memory requirement, the runtime will get substantially worse even if there is one mathematical operation can be planned by A*.

The root of the problem is the A* algorithm, since it is only partially suitable for such domain where goals are infinitely dimensional. A* is a fitting algorithm when x, y, z are the only dimensions however a task domain of 100 task has 100 pseudo-dimensions. For this reason our and GOAP’s implementation of A* only has a part of heuristic implemented. Weight is given to tasks through some mechanism; however distance to goal is not calculateable. A* is still preferred in such algorithms over DFS or BFS due

to its exploratory nature, since A* will always hone for the good heuristic path over bad heuristic nodes.

3.3. GHTN in Interactive Narrative

We have selected the “LaLa Land” universe as our domain and created our tasks thematically fitting with the universe. The main drive behind selecting “LaLa Land” is because it is a universe based on real world. Fantasy universes can also be applied, and studies such as Propp’s “Methodology of the Fairy Tale” can be utilized to create new tasks in many of the given fantasy universes.

3.3.1. Setting Initial States in Interactive Narrative

A world state is any variable that represent a state of the world such as; location of actors, belongings, and whether or not some important events have occurred.

The initial world state represents the beginning of the narrated world. The following world state would start the narrative where Sebastian has met Mia, is angry to the bar owner about the repertoire and has a concert tonight;

[metMia = true, angryAtRepertoire = true, concertTonight = false]

All the states that are missing in the above initial world state, but exist in the world state table are set to their values such as:

[miaGirlfriend = false, bandProblems = false … and others]

3.3.2. Setting Goal States in Interactive Narrative

Having a Goal as a state where Sebastian is Mia’s boyfriend and Sebastian’s band going through troubled time is expressed as:

[bandProblems = true, miaGirlfriend = true]

The missing states in the Goal are not set to their default value; they are indifferent for the algorithm. The algorithm does not guarantee every state will appear in the plan at some point. If every state is forced to appear in the plan, the outcome of our narrative

system would always give the same story but with different orderings of events and would disregard user interaction; this is not a valuable approach to narrative thus not the purpose of our system. Our purpose is to create explorative, continuous and non-repetitive narratives; therefore only related events in the behavior space should be included in the narrative. In our case, less is more.

We will now explore examples on tasks as Goal and categories as Goal. These 2 approaches will utilize the Goal as a bundle of world states system.

Method as a Goal, expresses the following requirement to the algorithm: “Create a story, where Sebastian opens his bar.” This Goal can be expressed as [BarOpening]. The algorithm registers this as a method due to the fact that there is no = sign in the Goal, and there is a task called BarOpening in the tasks. The system will now seek different paths that are ending with this task, and those different paths create the context of each question. If the system cannot find the required amount of different paths, the unconstructed questions will be hidden.

The other Goal input method is using a task category. When a task category is registered as a Goal, the system creates invisible world states to force members of the task category to be planned. Consider a task category called “Loss”, where the tagged methods are “LoseItem”, “FriendLost” and “LosePrevilege”. When the director inputs the category “Loss” as a Goal, the planner finds plans in the behavior space where these tasks occur at the very end of the plan, and direct player into experiencing one of these tasks. If a plan cannot be created for one of the Goals in the current world state, that Goal will be ignored to prevent a disconnected narrative. Each question will point to a different task in the category, if there are more plan-able Goals then the question count. If not, multiple questions may point to the same Goal through different paths.

3.3.3. Tasks and Methods

On top of the previous rules about tasks and methods, to satisfy requirements to create an interactive narrative there are additional design highlights.

The Interactive Planning algorithm starts plans from the task space root and plans only with the root tasks. Non-root tasks cannot be used in plans directly, however can be added as subtasks of root tasks.

Consider the case where “MiaRomantic” is a root task. Since this is a root task, the planner may use this task freely while planning. This will cause the narration to force the player to have romance with Mia all out of a sudden, without any backstory and create a disconnected story. To prevent discontinuity, “MiaRomantic” should be as Non-root task, so it should not be used in plans directly. Since “MiaRomantic” is a Non-root task, it requires some root task to work with.

To be in line with the universe, consider the task “GoCinema”. In our universe, people with affections can go to cinema to have a romantic time. To simulate a storyline in the narrative flow, “GoCinema” should be designed as follows;

```
Task(GoCinema)
    Method(inRelationShip = false, miaInvited = true)
        subtasks[miaRomantic]
    method(inRelationShip = false, sullyInvited = true)
        subtasks[sullyRomantic]
    method(bandInvited = true)
        subtasks[bandProblems = false]

Task(MiaRomantic)
    Method(miaImpressed = true)
        S-> "Attending to Mia's play was very important for
        her. She knows you care about her..."
        subtasks[miaGirlfriend = true, inRelationShip = true]
```

As seen in the figure, GoCinema will be called when MiaRomantic is planned and it will create a continuous narrative. The separation between root and Non-root tasks helps the designer tackle this problem with relative ease.

For interactive narrative purposes only, there is a unique tag called “milestone”. Milestone tasks are important tasks such as “GoCinema” or “CoffeeShop”. The reason these tasks are tagged as milestone is that they offer a good and bad outcome, which should not be spoiled while the questions are being constructed. “CoffeeShop” have two possible scenarios as meeting Mia or having a chill day, and “GoCinema” have three possible scenarios two of which are romance and last to strengthen band relationships.

This unique category “milestone” is utilized in the upcoming “Section 3.3.6: Interactive Planning”.

3.3.4. Scenarist

The Scenarist is the user responsible of the narrative flow. The Scenarist interacts with the algorithm through inputs such as setting the initial world state, setting the Goal and forcing the Goal.

Initial world state is the first input to the AI system; the system creates a world instance with the given set of states. This is the starting point of the narrative. Scenarist can set the starting location, starting possessions, starting ability and skills all at once at this point.

The Scenarist creates an initial world state with a bundle of states and their values. The states which are not mentioned in the initial world state bundle are set to their default values. The next input of the Scenarist comes directly after; the Goal. Scenarist prepares the Goal using the desired world states they want to have at the end of the execution.

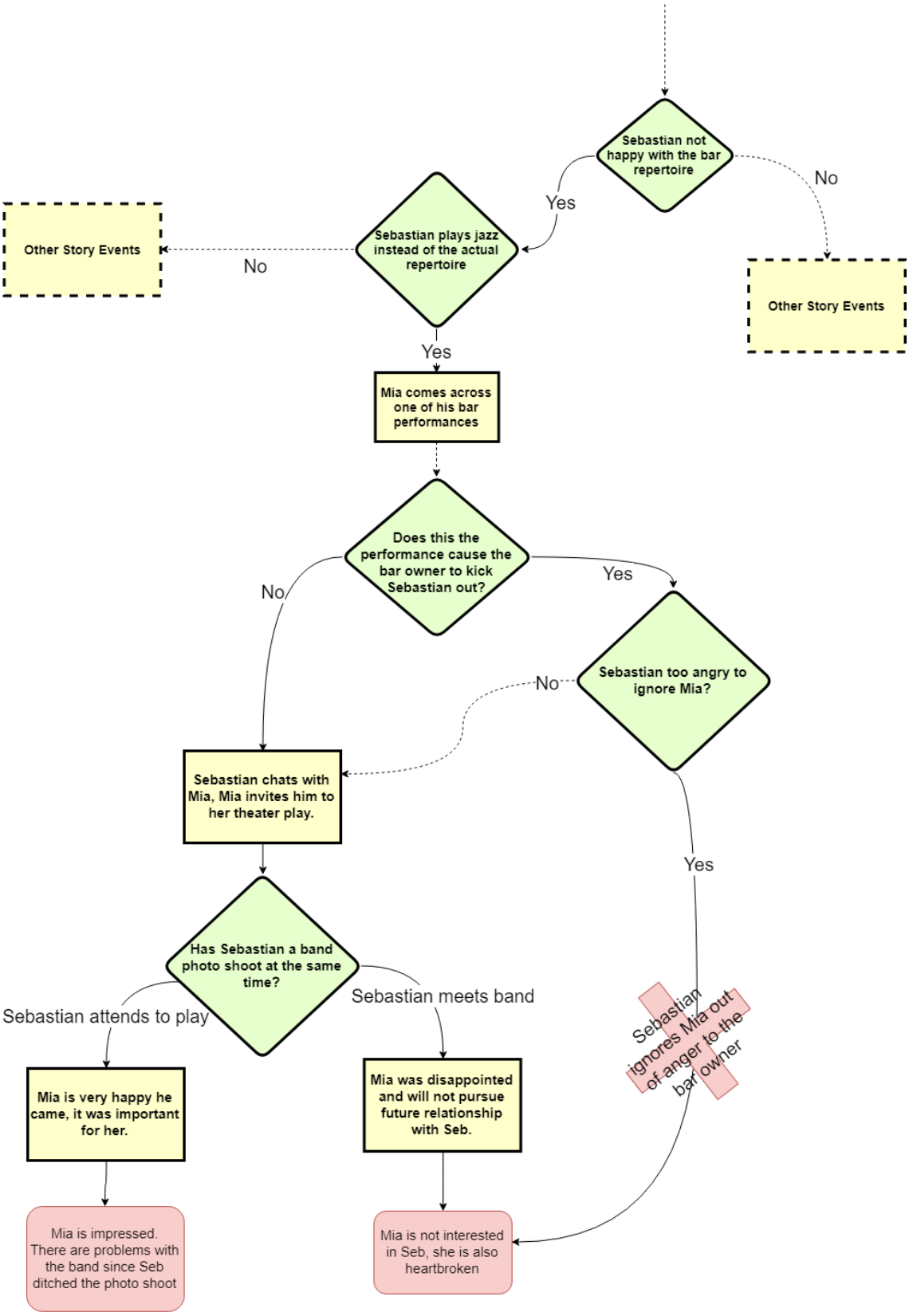

Figure 12: The ending story node takes the events that happened in the story into consideration and create an ending.

3.3.5. Scenarist and Task Preparation

When the Goal is entered, preparation algorithms are applied to favor the world into moving towards the Goal set by The Scenarist. Planning algorithms which bee-lines the user to the Goal will not work until The Scenarist inputs the “Force Goal” command. With the force Goal command, the algorithm allows Scenarist to decide approximately when to end the narration. Having two modes of operation where the algorithm silently pushes you to the Goal and the algorithm forces you to the Goal creates the illusion of freedom, adds replayability value while maintaining the guarantee that the user will end up in the Goal State.

After the Scenarist enters the Goal for the first time, preparation algorithm runs to move related tasks higher in the order of all the behavioral tasks; therefore passively creates direction in the story. Afterwards, the methods that are contradicting with the goal are disabled to prevent a stuck state; where the Goal can never be reached using the tasks in the behavioral space.

3.3.6. Interactive Planning (Asking Questions)

The algorithmic background being discussed in Section 3.1.10, we will now explore how to formulate question in the interactive narrative domain.

The Interactive Planning algorithm returns N paths, where N is the number of questions at every step. Between the N paths, the longest common subsequence from the start is calculated for each plan, 1 more task is added to the subsequence and these subsequences are formatted as interaction strings and can now be prompted.

Script strings are denoted by “S” and question strings are denoted by “Q”;

```
Task(CoffeeShop)
    method(miaMet = false)
        Q-> "Prepare your car and leave for the
        Coffee Shop to have a chill day."
        S-> "While on your way to the Coffee Shop,
        someone cut to your lane. You almost had
        an accident. The driving lady shouted at
        you as if it was your fault"
        subtasks(metMia = true)
```

Consider N is 3, so the Interactive Planning algorithm returns these 3 plans where every letter corresponds to a task;

	First Milestone	Script String	Question String
ABC		ABC	
DEF		DEF	
GHIJ		GHIJ	

Table 1: Every plan is different and there are no milestones

	First Milestone	Script String	Question String
ABC	A		A
DEF		DEF	
GHIJ	J	GHI	J

Table 2: Every plan is different but there are milestones; (Assume A, B, J are milestones)

	Noncommon	First Milestone	Script String	Question String
ABCDE	ABCD		ABCD	
ABCGP	ABCG		ABCG	
AHJKP	AH		AH	

Table 3: Plans overlap and there are no milestones

	Noncommon	First Milestone	Script String	Question String
ABCDE	ABCD	B	A	B
ABCGP	ABCG	B	A	B
AHJKP	AH		AH	

Table 4: Plans overlap and there are milestones.

ABCDE and ABCGP plans should not be printed as is to not spoil the outcome of task “B” in the script text, “B” should be in question text. Since “B” is a milestone task, both these questions will print up to A.script + B.question. However this will cause two questions to be exactly the same, which is an unwanted behavior. In an outcome like this, we ask the planner to come up with a new plan while blocking the “AB” and “AHJKP” plans. The blocking operation is done by modifying the heuristic on the selected plans last task locally (specifically on that instance) to max possible heuristic value; thus the planner will not evaluate those paths further. If the planner can come up with an alternative plan, the mentioned algorithm is reapplied, if there are no alternative plans the 3rd question is omitted.

3.4. Further Research

3.4.1. Bidirectional Search

Instead of a unidirectional A* algorithm implementation, Bidirectional Search can be applied to reduce complexity. In Bidirectional Search, two search algorithms start simultaneously, from goal to initial state and from initial state to goal. The search terminates when the fringes of these two algorithms meet; if they expand to the same world state.

While the goal-to-initial search will work the same as the unidirectional A*, the initial-to-goal algorithm will work differently. In theory, since the initial state includes all the possible world states that are defined in the behavioral space, the search starting from the initial world state will be more exploratory in nature with respect to the goal-to-initial search where the starting states are severely limited. Since the exploratory nature expand the fringe of the initial-to-goal search, the goal-to-initial search will have more points for termination; which will cause in less node exploration in total. Even though the time cost of the algorithm is better than its unidirectional counterpart, we believe it can be further improved by testing weighted uses on the goal-to-initial and initial-to-goal searches rather than running them on equal resources.

Instead of using weights to further improve bidirectional search in our domain, a state-space representation can be studied on the system to further improve the general search methodology. In his book Planning Algorithms(LaValle, 2006) chapter 2, LaValle discusses how STRIPS-like models can be converted to State-Space Representation, and logic-based planning methods can be applied through creating planning graphs.

3.4.2. Novelty Pruning

Novelty Pruning is a method proposed in “Fast and Diverse Narrative Planning through Novelty Pruning” (Farrell and Ware, 2016) to prune the non-novelty nodes in the search space. The algorithm is implemented on “Glaive: A state-space narrative planner supporting intentionality and conflict” (Ware and Young, 2014) and different algorithms and benchmarked in the proposing paper. A node is removed from the node space if it not effective neither on the current state nor on any of the planner’s ongoing planning states. By doing so, the amount of non-novelty nodes will be reduced, improving the planning speed. It is reported that “Glaive with novelty pruning is up to 107.3 times faster than Glaive without novelty pruning”.

Both Glaive and our algorithm being A* based, a similar approach to Novelty Pruning is also applicable in our algorithm. The reduced search space may greatly reduce the total expansion of A* through disabling the nodes deemed to be not novelty. Novelty Pruning may also help us tackle the problems we have with integers, as discussed in “Section 3.2.5: A* and The Integer Problems”.

Novelty Pruning can also bring different advantages to our data preparation system discussed in “Section 3.1.4: Preparing Behavior Space for Goals”. While our algorithm reduces the unrelated tasks in the hierarchy, Novelty Pruning completely eliminates the tasks which are deemed non-novelty. Novelty Pruning being a much more aggressive approach, will lead to better outcomes in large domains when paired with heuristic search techniques, where many alternative tasks may exist in the domain.

Motivation Pruning is another technique discussed in this paper, designed to work alongside Novelty Pruning. Since Glaive is designed for Interactive Narrative, it requires finding explanations to steps in the plans the algorithm create. When an explanation cannot be created by a node, the node is pruned since it is certain that this node can never lead to a solution.

Both of these algorithms strengthen the quality of the narrative created by the planner. While we lack such pruning algorithms, we partially fill these requirements through preparing the data when a new goal is assigned to the system. Our algorithms disable contradicting methods; which is similar to Motivation Pruning, and reorders the task network depending on the given goal, similar to Novelty Pruning. Both the Motivation

Pruning and Novelty Pruning are much aggressive than our algorithms and further study is required in order to apply such aggressive algorithms to our systems better.

3.4.3. Interactive Narrative Quality

Three main features discussed by "Fast and Diverse Narrative Planning through Novelty Pruning" (Farrell and Ware, 2016) to ensure quality plans are;

Build branching stories while maintaining user agency

In our algorithm when the Scenarist forces the user to the ending, every step is taken by the user and the steps are guaranteed to end at the goal (assuming there are valid plans to the goal). Due to our planning algorithm, our system searches the task space to come up with choices with no overlap at the beginning. However in our system we don't have free-will of the user; the choices of the user are created beforehand in planning and are presented as alternatives. Our system can be improved to have a dynamic number of questions asked, instead of a static number; depending on the number of possible plans possible in the domain.

Allow the human author to choose from possible goals

The human author in our case is called the Scenarist. In our system the Scenarist is completely free in picking goals from the list of world states. This is a two edged sword; while the Scenarist is completely free in choosing the goals, we require the Scenarist to have absolute knowledge on the domain in order to understand the world states to come up with a goal. The Scenarist system can be improved with quality of life features such as giving a dependency graph on the world states to explain the underlying system in the domain.

Every goal set is not guaranteed to be possible; the goal can be directly or indirectly conflicting with each other. As discussed in "Section 3.4.4: Guarantee of Finding a Plan"; the planner is guaranteed to find a plan with enough resources. To avoid impossible goal sets, the planner can be utilized after a goal set is registered to the system to check if the goal set is possible to satisfy.

ISIF distance metrics to make sure created plans are unique

ISIF, Important-Step Intention-Frame algorithm is a distance metric that compares two plans. The summaries of intention frames are calculated with the following quadruple <c,g,m,f> where c is the character, g is the goal, m is the task that motivates toward the goal, and f is the final satisfying step that accomplishes the goal g.

The ISIF distance will be closer to 0 for plans which have the same motivating step and the same achieving step. In other words, the most differentiating elements of a plan are the initial and the last step of the plan.

In our algorithm we are creating questions when a milestone task is encountered. Therefore our algorithm makes sure that the first real important point in the story is recognized by the system and our other plans are calculated in a way that ignores the task related with the question created at that point, so the other plans generated by the algorithm will not consider this milestone task. Therefore we guarantee that if there exists multiple plans to the goal; the algorithm will never pick the already selected initial step for its plan again.

The other important parameter in the ISIF distance is the final task which satisfies the goal. In our algorithm we have no subsystems to ensure/force a difference at the last step of the plan. In order to have a greater ISIF distance and therefore have better diversity in our plans, our algorithm can be expanded with subsystems which prefer unused tasks to be the final satisfying step instead of already used tasks. Such subsystem will be a direct improvement for ISIF distance metrics and our plan quality.

3.4.4. Guarantee of Finding a Plan

Having an exhaustive search algorithm guarantees finding a plan given enough time and memory for the planner. Algorithms such as BFS or DFS will visit every possible node and their combinations in the task space to come up with a plan towards the goal. The following table explains Farrell and Ware's comparison of BFS and Glaive on Novelty Pruning, however the table is modified to only include non-Novelty Pruning columns.

Problem		Planner	
		BFS	Glaive
Heist	solution	out of time	out of time
	visited	19,266,167	1,046,098
	generated	365,567,309	22,723,488
	pruned	117,503,623	6,267,374
	time (ms)	3,633,640	3,600,002
Space	solution	2 steps	2 steps
	visited	6	3
	generated	56	24
	pruned	14	6
	time (ms)	< 1	< 1
Fantasy	solution	6 steps	6 steps
	visited	55,394	20,221
	generated	818,894	338,248
	pruned	135,487	46,595
	time (ms)	3,233	151,724
Western	solution	5 steps	6 steps
	visited	14,879	176,028
	generated	414,165	4,341,287
	pruned	132,605	799,078
	time (ms)	2,366	695,648
Ark	solution	7 steps	7 steps
	visited	4,132	186
	generated	26,601	950
	pruned	5,039	178
	time (ms)	127	22
BLP-Die	solution	7 steps	7 steps
	visited	1,271,055	1,250,936
	generated	23,472,945	21,924,886
	pruned	16,460,269	15,479,744
	time (ms)	1,662,262	2,886,313
BLP-Win	solution	10 steps	11 steps
	visited	1,271,055	1,057,967
	generated	23,472,945	18,416,493
	pruned	16,460,269	13,040,600
	time (ms)	1,663,905	2,440,509
Life	solution	11 steps	12 steps
	visited	37,144	97,191
	generated	225,850	613,110
	pruned	117,101	407,505
	time (ms)	5,066	39,999

Table 5: Glaive & BFS Algorithm comparison in different domains

As seen in the table, both Glaive (A* heuristic based) and BFS can both come with solutions in the given task spaces. These algorithms were limited with 100GB of RAM, 3.5 GHz Intel Xeon processor and 1 hour time.

The difference between these algorithms and our algorithm is the domain. While Glaive works with nodes and plan a pathway through the nodes, our algorithm works on a Hierarchical Task Networks, which is composed of multiple 1 depth networks of nodes. Our preparation algorithm on “Section 3.1.8: Preparing Behavior Space for Planning” dissolves/flattens the network into independent nodes therefore our A* planner can work on the system. Via the preparation algorithm, the planning side of our domain becomes identical with Glaive’s domain. There is no correlation between nodes, every node has preconditions and post-conditions, every node are composed of a single method therefore have no unseen preconditions.

For these reasons, we can safely say that a Glaive-like A* heuristic planner algorithm, as well as BFS would work on our domain successfully. Our implementation of A* is simpler than Glaive’s, which lacks or only satisfies some requirements to a certain degree; mostly related with pruning algorithms explained in the previous “Section: 3.4.2: Novelty Pruning”. Future research on our planner would include implementation of bidirectional search and similar higher performance search algorithms, or improving the state-space representation in our system to prepare the system for logic-based search methods as explained in “Section 3.4.1: Bidirectional Search”.

3.5. Full Algorithm Overview in a Test Environment

For simplicity, questions are not written at all and post selection texts are not written in full form. World state changes are visible to make tracing through the example possible.

The Q line is where the user is asked to do a selection between 3 choices and it is followed by how those selections affect the world states. The user's choice is displayed as the underlined question. The planner interaction starts at the start of day 3. Before planner starts, the choices are given solely through preprocessed orderings of the tasks.

```
Goal:
bandProblems = true
miaGirlfriend = true

Ordering before preprocessing:
Concert
MeetBand
PianoPractice
BarConcert
JobSearch
AttendPlay
CoffeeShop
Club
StayHome
Groceries
Sports
LoneNight
NightWalk

Ordering after preprocessing:
PianoPractice ~ Jazz takes Mia's interest
CoffeeShop ~ Meet Mia's friend
AttendPlay ~ Important for Mia
Groceries ~ Meet Mia's friend
Concert
MeetBand
BarConcert
JobSearch
Club
StayHome
Sports
LoneNight
NightWalk
```

The narrative starts;

Q Coffee Shop, Stay Home, Groceries

- metMia = true
- "While on your way to the Coffee Shop, someone cut to your lane. You almost had an accident. The driving lady shouted at you as if it was your fault..."

Q Stay Home, Meet Band, Sports

- photoShootOnDay4 = true
- "Meeting with the band, you plan a photoshoot on 4th day for publicity."

Q TRIGGER(nightTime) [time == 2]

- night = True

Q Lone Night, Nightwalk, Club

- "Seb doesn't feel much energetic, he spends a lonely night by the television"

Q TRIGGER(dayTime) [time == 3]

- day++ //day = 1
- time = 0
- night = false
- concertTonight = false

Q TRIGGER(BarConcertNight) [day % 7 == 1 or day % 7 == 2]

- concertTonight = true
- "Tonight Seb has to be at work at bar."

Q Coffee Shop, Stay Home, Groceries

- "There were nothing interesting happening at the coffee shop. You get yourself a drink and head back home"

Q Stay Home, Meet Band, Piano Practice

- angryAtRepertoire = true
- "Due to the upcoming bar concert tonight, you wanted to practice. The repertoire is too boring you think to yourself. You can't express your feelings with classical.."

Q TRIGGER(nightTime) [time == 3]

- night = True

Q Concert Night, Nightwalk, Club

- lastChanceAtBar = true
- "Once again the owner was mad due to your jazz piano improvisations and not following the repertoire. Last chance he says, getting tired of you."

Q TRIGGER(dayTime) [time == 3]

- day++ //day = 2
- time = 0
- concertTonight = false

Through goal forcing, the planner starts working at this point on day 3.

At this stage in the algorithm, we expect the planner to return us three plans from the initial world state to the goal, to prompt questions to the user. For simplicity; we simulate one plan, this plan is not the lowest cost path, nor is it guaranteed to be selected by the planner. With this example our purpose is to show that the planner works as a whole system, and plans can be generated using the algorithm.

```
Initial World State:
metMia = true
photoshootOnDay4 = true
angryAtRepertoire = true
lastChanceAtBar = true
concertTonight = false

Goal:
bandProblems = true
miaGirlfriend = true

Planner World State:
miaGirlfriend = true <=
bandProblems = true
```

From goal, the planners world state is generated. The currently iterated state is miaGirlfriend = true.

The planner looks at the tasks to find one where miaGirlfriend = true happens.

There are two methods, MiaRomantic:Method1(cinema with good reputation with Mia) and MiaRomantic:Method3(protect Mia from thugs outside club)

The planner selects the first method; for the sake of simplicity we are ignoring the fact that Method3 may have a better heuristic. How A* works can be better explained decoupled from this example.

MiaRomantic:Method1

```
Task(MiaRoantic)
    Method(miaImpressed = true)
        S-> "Attending to Mia's play was very important for
        her. She knows you care about her..."
        subtasks[miaGirlfriend = true, inRelationShip = true]
```

Therefore the planner's world state should become:

```
.miaImpressed = true
!miaGirlfriend = true <=
%inRelationShip = true
```

Even though this is a valid plan step, this is not executable by the user since MiaRomantic is not a root task, it is a non-root task. The user can only be prompted with root tasks. At this point the algorithm must find a root task, which calls MiaRomantic.

Looking at the tasks, the algorithm finds the root task GoCinema, which calls MiaRomantic.

```
Task(GoCinema)
    Method(inRelationShip = false, miaInvited = true)
        subtasks[miaRomantic]
    method(inRelationShip = false, sullyInvited = true)
        subtasks[sullyRomantic]
    method(bandInvited = true)
        subtasks[bandProblems = false]
```

The GoCinema task above is the original task in our system. On preparation for planning phase the system prepares the methods to be independent from each other. Following is the actually prepared GoCinema method utilized by the planner.

```
Task(GoCinema)
    1 Method(inRelationShip = false, miaInvited = true)
        subtasks[miaRomantic]
    2.1 method(inRelationShip = false, sullyInvited = true, inRelationShip = true)
        subtasks[sullyRomantic]
    2.2 method(inRelationShip = false, sullyInvited = true, miaInvited = false)
        subtasks[sullyRomantic]
    3.1 method(bandInvited = true, inRelationShip = true, inRelationShip = true)
        subtasks[bandProblems = false]
    3.2 method(bandInvited = true, inRelationShip = true, sullyInvited = true)
        subtasks[bandProblems = false]
    3.3 method(bandInvited = true, miaInvited = true, inRelationShip = false)
        subtasks[bandProblems = false]
    3.4 method(bandInvited = true, miaInvited = false, sullyInvited = false)
        subtasks[bandProblems = false]
```

The planner algorithm selects GoCinema:1 and the partial plan currently is;

```
Plan:
GoCinema
```

The preconditions of GoCinema are added on top of the planner's world state and the following world state is created.

```
Planner world states:
.inRelationShip = false
.miaInvited = true
miaImpressed = true
!miaGirlfriend = true <=
%inRelationShip = true
```

The iteration now moves to the first state, inRelationship = false.

Since this state matches with initial world state, it is satisfied, symbol "+" is added to the state.

The iteration moves to the next state, miaInvited = true

```
Planner world states:
+inRelationShip = false
.miaInvited = true  <=
miaImpressed = true
!miaGirlfriend = true
%inRelationShip = true
```

To solve miaInvited, the planner plans the task CoffeeShop;

```
Task(CoffeeShop)
    method(attendedPlay = true)
        S-> "At the coffee shop Seb sees Mia's friend from
        the play. Friend tells Seb to maybe invite Mia to
        cinema. You do so. Mia accepts."
        subtasks(miaInvited = true)
    method(miaMet = false)
        Q-> "Prepare your car and leave for the Coffee
        Shop to have a chill day."
        S-> "While on your way to the Coffee Shop, someone
        cut to your lane. You almost had an accident. The
        driving lady shouted at you as if it was your fault"
        subtasks(metMia = true)
```

CoffeeShop:1 satisfies the planners world state "miaInvited = true", therefore this method is added to the plan.

```
Plan:
GoCinema
CoffeeShop


Planner world states:
+inRelationShip = false
!miaInvited = true  <=
miaImpressed = true
!miaGirlfriend = true
%inRelationShip = true
```

The iterator moves to “miaImpressed = true”.

The planner algorithm searches for tasks that satisfy the state “miaImpressed = true”.

```
Task(AttendPlay)
    method(photoshootOnDay4 = false, invitedToMiaPlay = true)
        S-> "You attended Mia's play. You start think to yourself highly
        of her"
        subtasks[miaImpressed = true, likeMia = true]
    method(photoshootOnDay4 = true, invitedToMiaPlay = true)
        S-> "You attended Mia's play. You start think to yourself highly
        of her. But your band will not be happy about your silent
        disappearance"
        subtasks[miaImpressed = true, likeMia = true, bandProblems = true]
```

The algorithm sees the counterpart where the methods were prepared at the start, but the prepared methods are identically same with the original one since method 2 is called, and method 2 has a differentiating state with method 1. These methods are guaranteed to stay as is after preparation.

The algorithm picks QuidditchCup:2 and the following world state is created.

```
Planner world states:
+inRelationShip = false
+miaInvited = true
.photoshootOnDay4 = true
.invitedToMiaPlay = true
!miaImpressed = true    <=
%likeMia = true
%bandProblems = true
!miaGirlfriend = true
%inRelationShip = true
```

The iterator moves to "photoshootOnDay4 = true". This state is already satisfied with the starting world state. Iteration moves to “.invitedToMiaPlay = true”.

```
Planner world states:
+inRelationShip = false
+miaInvited = true
+photoshootOnDay4 = true
.invitedToMiaPlay = true    <=
!miaImpressed = true
%likeMia = true
%bandProblems = true
!miaGirlfriend = true
%inRelationShip = true
```

The planner finds this world state in the task MiaHearsYouPlay. However since this is not a root task, the planner plans a root task which calls MiaHearsYouPlay.

```
Task(MiaHearsYouPlay)
    method(angryAtRepertoire = true)
        S-> "Loving your performance at the bar
        tonight, Mia invites you to her play."
        subtasks[invitedToMiaPlay = true]
```

The planner uses BarConcert since it calls MiaHearsYouPlay.

```
Task(BarConcert)
    method(angryAtRepertoire = true, lastChanceAtBar = true)
        S-> "Seb gets lost in emotion while playing and
        starts to play jazz again. The angry manager tells
        him that he is fired. On his way out Seb sees a
        lady waiting for him."
        subtasks[haveJob = false, MiaHearsYouPlay]
```

The invitedToMiaPlay = true" state is accomplished. The preconditions and sister subtasks are added. The next iteration is “angryAtRepertoire = true”.

```
Plan:
GoCinema
CoffeeShop
AttendPlay
BarConcert

Planner world states:
+inRelationShip = false
+miaInvited = true
+photoshootOnDay4 = true
.angryAtRepertoire = true    <=
.lastChanceAtBar = true
%haveJob = false
!invitedToMiaPlay = true
!miaImpressed = true
%likeMia = true
%bandProblems = true
!miaGirlfriend = true
%inRelationShip = true
```

To satisfy ".angryAtRepertoire = true" the planner finds the task PianoPractice, first method.

```
Task(PianoPractice)
    method(haveJob = true)
        S-> "Due to the upcoming bar concert
        tonight, you wanted to practice. The
        repertoire is too boring you think to
        yourself. You can't express your feelings
        with classical.."
        subtasks[angryAtRepertoire = true]
    method(haveJob = false)
        S-> "Practicing jazz soothes Seb. He has
        some very likeable resonance stuck in his
        head. Feeling intrigued, Seb would love
        to talk about this with his band mates.
        subtasks[shouldContactBandMates = true]
```

The task PianoPractice satisfies the world state "angryAtRepertoire =true" and the following state & plan are created.

```
Planner world states:
+inRelationShip = false
+miaInvited = true
+photoshootOnDay4 = true
+angryAtRepertoire = true
.lastChanceAtBar = true <=
%haveJob = false
!invitedToMiaPlay = true
!miaImpressed = true
%likeMia = true
%bandProblems = true
!miaGirlfriend = true
%inRelationShip = true


Plan:
GoCinema
CoffeeShop
AttendPlay
BarConcert
PianoPractice
```

Iterator is on ". lastChanceAtBar = true" which is satisfied in the world state, symbol "+" is given.

```
Planner world states:
+inRelationShip = false
+miaInvited = true
+photoshootOnDay4 = true
+angryAtRepertoire = true
+lastChanceAtBar = true <=
%haveJob = false
!invitedToMiaPlay = true
!miaImpressed = true
%likeMia = true
%bandProblems = true
!miaGirlfriend = true
%inRelationShip = true
```

Plan is successful since there are no tasks left without a satisfied symbol!

The current plan is final!

```
Final Plan:
GoCinema
CoffeeShop
AttendPlay
BarConcert
PianoPractice
```

Following is a narrative flow assuming this plan is followed from start to end.

```
!!PLANNER

Trigger ~ BarConcertNight
**Day3
*BandMeeting ~ Filler: Seb meets with band  at
the studio for practice
*PianoPractice ~ Planned: Seb hates his repertoire
Trigger ~ Night Time
*BarConcert ~ Planned: Seb gets fired and Mia
invites him to her play

**Day4
*JobSearch ~ Filler: Being fired, Seb looks for
musician jobs nearby.
*AttendPlay ~ Planned: Seb attends to play but
misses photoshoot of band
*Trigger ~ Night Time
BandMeeting ~ Filler: Band calls Seb, stunned by
his irresponsible actions

**Day5
*CoffeeShop ~ Planned: Mia's friend tells Seb to
invite Mia, Seb calls her to cinema, she accepts.
*GoCinema ~ Planned: Seb and Mia are on a date
~Reached Goal~
```

In this planning run, the algorithm wanted to create a plan which satisfies the following two requirements: Mia is Seb's girlfriend and disagreements occurred in Seb's band. The planning starts on the start of 3rd day and guides Seb through the goals. This example illustrates a single plan and additional plans will also be created alongside this one to give multiple choices to the user.

Bibliography

[1] Faria, A. J., et al. "Developments in business gaming a review of the past 40 years." Simulation & Gaming (2009).

[2] Hill, R., et al. "Virtual humans in the mission rehearsal." kunstliche Intelligenz (2003).

[3] Kato, P. M. "Video games in health care: Closing the gap." Review of General Psychology (2010).

[4] Magerko, B., B. Stensrud and L. Holt. "Bringing the schoolhouse inside the box-a tool for engaging, individualized training." 25th Army Science, 2006.

[5] Riedl, M. O., et al. "Dynamic experience management in virtual worlds for entertainment, education, and training." International Transactions on Systems Science and Applications (2008).

[6] Zee, D.J. Van Der, B. Holkenborg and S. Robinson. "Conceptual modeling for simulation-based serious gaming." Decision Support Systems (2012).

[7] Faria, A. J., et al. "Developments in business gaming a review of the past 40 years." *Simulation & Gaming* (2009).

[8] Hill, R., et al. "Virtual humans in the mission rehearsal." *kunstliche Intelligenz* (2003).

[9] Kato, P. M. "Video games in health care: Closing the gap." *Review of General Psychology* (2010).

[10] Magerko, B., B. Stensrud and L. Holt. "Bringing the schoolhouse inside the box-a tool for engaging, individualized training." 25th Army Science, 2006.

[11] Riedl, M. O., et al. "Dynamic experience management in virtual worlds for entertainment, education, and training." *International Transactions on Systems Science and Applications* (2008).

[12] Zee, D.J. Van Der, B. Holkenborg and S. Robinson. "Conceptual modeling for simulation-based serious gaming." *Decision Support Systems* (2012).

[13] Humphreys, T. "Exploring HTN Planners through Example". Game AI Pro, 3, 149-167. (2017).

[14] Straatman, R., et al. "Hierarchical AI for Multiplayer Bots in Killzone 3". Game AI Pro, 3, 377-390. (2017).

[15] LaValle, S. M. Planning algorithms. New York (NY): Cambridge University Press. Chapter 2 Discrete Planning (2014).

[16] Mark, D. AI Architectures: A Culinary Guide. Retrieved from http://intrinsicalgorithm.com/IAonAI/2012/11/ai-architectures-a-culinary-guide-gdmag-article/ (2012, November 1).

[17] Farrell, R. and Ware, S. "Fast and Diverse Narrative Planning through Novelty Pruning", 12th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (2016).

[18] Orkin, J. "Agent Architecture Considerations for Real-Time Planning in Games", Monolith Productions. (2004).

[19] Orkin, J. "Applying Goal-Oriented Action Planning to Games", Monolith Productions. (2003).

[20] Orkin, J. "Three States and a Plan: The AI of FEAR", Monolith Productions / MIT Media Lab, Cognative Machines Group. (2003).

[21] Isla, D. "Handling Complexity in the Halo 2 AI", GDC 2005. (2005).

[22] Kelly, JP. "Planning with Hiearchical Task Networks in Video Games", Australian National University. (2007).