HTN and Behaviour Trees for improved coaching AI in RTS games

James Johnston, student ID:100121625
J.Johnston2@unimail.derby.ac.uk

Abstract—This paper presents a study and prototype of the significance of Hierarchical task network systems combined with a layered behaviour tree in games programming. Highlighting the history of both Hierarchical task networks and behaviour trees amongst other prevalent techniques before outlining and evaluating a hybrid system that utilises the benefits of both. This system is then evaluated via the use of a prototype game Janus (jey-nuhs) and potential improvements are discussed for future development and application.

1 INTRODUCTION

Video games have evolved extensively but due to the development time often available for games and the emphasis on reduced cost and player enjoyment often strategies are used to create the effects of more intelligent solutions in order to reduce development costs and time, with this illusions of intelligence occasionally falling short of human expectations and damaging player experience.

However as games grow into larger and more complex systems with larger teams and resources considerations are being made to make better use of AI and planning systems commonly used in other areas of computing.

The goal of this project is to outline the recent advances of video games to utilise more advanced AI systems and evaluate their strengths and weaknesses. Before progressing on to propose a theoretical hybrid planning solution that can be quickly implemented and expanded to solve many in game tasks. Whilst also providing a realistic and believable player experience and not sacrificing development time or incurring extra costs.

2 PLANNING IN MODERN GAMES

Before outlining the system proposed and examining the test case first a brief overview of current planning techniques used in video games as well as techniques that are in use in traditional computing situations in order to provide a base level of knowledge and context.

2.1 Key Theories and terms

It is important to understand the terms used traditionally in AI research and the techniques used for this project. Many of these terms however are kept specific to the areas focused on by this project so this may not represent a full description of these fields.

2.1.1 Agents

An Agent is one of the core principles of AI and refers to any unique entity that is capable of making decisions and carrying out actions [1, pg. 4 - 5]. These Agents can exist in varying degrees of complexity from simple automata all the way to fully responsive AI that learn about their environment dynamically. Agents can start with information about the world in which they exist or with no knowledge or anywhere in between. Agents can make either intelligent decisions based on knowledge available or may simply act pseudo randomly however the second would not be considered intelligent in and of it’s self. It is possible for agents to incorporate an element of random in order to discover new strategies or to overcome predictable natures that players may feel are uninteresting.

2.1.2 Hierarchical Task Networks

Hierarchical Task Networks (HTNs) is one branch of automated planning technology and revolves around the premise of three main types of tasks used to define and then reach any goal state [2]. The basic tasks/methods are those that can actually be executed by an agent, such as moving an agent forward 5 meters e.t.c. These can then be combined together into compound tasks that involve completing one task after the completion of it’s predecessors. And finally it makes use of goal tasks which represent the state that the Agent or Agents are attempting to achieve and are defined by a series of conditions that must be made true. Each task may have a set of prerequisites or requirements as well as effects or outcomes of a primitive or compound task being performed. How these tasks are decided depends on the context and some systems will allow the agents to constantly check for new better plans and some will simply rely on staying in the current plan until it is completed or failed. Tasks are chosen that will cause the goal task to become true with either the fewest number of actions or time taken to perform actions depending
on the requirements of the system. There are several established libraries that implement HTN including SHOP2 [3] and JavaFF [4] that have been developed to provide domain independent planners for any planning problem.

2.1.3 STRIPS Planning
STRIPS (Stanford Research Institute Problem Solver) planning functions in a similar manner to HTN, it defines a list of available actions and their preconditions as well as a goal state that should be achieved [5] [6]. It also defines an initial state for the world to begin in and a post condition states for all actions defining what will happen as a result. STRIPS plan construction relies on reaching a state in which all of the goal state conditions are true. Then it is a case of finding a chain of tasks that transition from the initial state to the end state. STRIPS planning does not guarantee the most optimal series of actions is performed in order and depending on the requirements of the planner some planners may choose to use the first viable plan in order to reduce the computation time spent. This may be preferable in video games where often only a limited amount of planning time is available to agents and quick plans may be preferable.

2.1.4 Finite State Machines
Finite State Machines are a form of AI decision making often used in video games for performing simple tasks and rely on define all states that an entity can possibly be in and actions available to perform in these states [7, pg. 256-257]. A state could contain as little information as to whether an agent has an item and if so it may perform a series of actions with that item. FSMs can be used for opponent AI in video games and can produce fixed predictable responses for an AI such as flee when health is low etc. This AI is one commonly relied on and used in video game development as it provides quick logical control over agents. It is important to note that transitioning from one state to another is only defined by the transitions available in the current state and may not be shared in all states [8, pg. 237]. This means that while it may logically be possible to change state to another state in the real world if the state change is not available to an agent it cannot perform it and would have to go through an intermittent state first.

2.1.5 Behaviour trees
Behaviour Trees are similar to finite state machines but define a tree of actions with all decision making starting at the highest level of the tree and then following the rules of each node in the tree to decide the next node to examine before determining an action once reaching a leaf node (a node with no children, in this case representing a decision to make) [9]. A behaviour tree should not store any information on the state of an agent but should determine this information as it is needed. Where as a finite state machine locks an agent into a state such as In combat and then defines a limited number of ways it can transition to another state normally through actions. They represent a simple solution to many planning problems, they rely on simple condition management and can be quick to implement but difficult to maintain once the behaviours available grow significantly. Behaviour tree’s can be structured to make use of modular behaviours that can be accessed from several branches of a tree, this is attractive to developers as it allows for functionality to be reused. They are quite common for solving problems in games as the modular design allows reuse of behaviour at different levels or situations, and suitable control is left to designers and developers on actions to take in special situations allowing overrides of logical behaviour for the sake of player enjoyment.

2.2 Modern games and planning technology
2.2.1 Unreal Tournament
Planning techniques have been applied to several video environments by academic teams including the work of Hai Hoang et al. [10]. This research looked at the theory of applying a HTN planning system to a squad based AI for the game Unreal Tournament 2004 and subsequently develops an improved TMK (Task method knowledge) system for managing the squad AI this presents intelligent AI that better mimic players by combining there efforts into more believable strategies.

2.2.2 F.E.A.R
The 2005 game F.E.A.R. used STRIPS based planning system and was one of the first games to develop an improved AI system that felt engaging to players and was practical to develop [11]. Whilst some techniques had been implemented before F.E.A.R. achieved critical acclaim for it’s work and helped to gain traction for intelligent and complete AI solutions. It made full use of STRIPS planning to achieve goals such as attacking and killing the player, and by being combined effectively with a comprehensive animation system it was possible to create behaviour that appeared more complex then it was such as leaping in to cover simply being composed of a move to task and an appropriate animation. F.E.A.R utilises only 3 states in which an AI can exist and by combining this with the intelligent animation system actions such as flipping a table to create cover can seem like tactical choices for AI but are simply a series of interactive objects and connected animations.

2.2.3 Halo 3
In 2008 the developers of Halo 3 [12] explained their AI implementation at the Game Developers Conference in San Francisco [13]. Here they outlined their system for designer driven tasks in an environment. These tasks could be as simple as defending a series of locations and the system then assigns squads of AI soldiers to each task based on priority and the squad specific information. The Tasks and enemy AI existed at a simpler level however
and the complex and believable behaviours came from the scripted squad goals such as defending and then falling back if they are overrun. This system is very interesting and unique as it allows good levels of control and if a squad is defeated then another squad can be pushed up from less important tasks to perform the newly vacant task much like a real world combat situation. It provides a good level of customisation however it also requires a heavy amount of scripting beforehand and treats every environment as a separate entity that must be scripted by a team of designers, but reduces the level of complication of AI calculations and produces good results for this context. It may not be suitable for a Hierarchical set of problems where there are multiple steps with interdependent prerequisites and results that require more complex plan construction.

2.2.4 Killzone 2

Killzone 2 [14] used a full HTN planning system for it’s AI opponents. They implemented a full HTN planning system that evaluated the state space of the battlefield and constructed plans [15]. It was also capable of aborting plans pre-emptively when a better plan became available or the current plan become infeasible. This shows that complex games including large amounts of physics calculations and high quality art assets are still able to carry out full HTN planning whilst maintaining an acceptable frame rate. However it is important to note the size of the project team working on Killzone as well as the funding and time taken to develop this system, which may not be available on other projects that still require planning systems. It is also possible that the system developed for killzone was more complex then required to provide an engaging experience to the player which is an important goal of any planning system in video game production.

3 JANUS PLANNING PROTOTYPE

After looking at these systems for planning within an AI context we propose a simplified version of a combined Behaviour tree/HTN planning system that can be used to solve a relatively simple task whilst maintaining it’s extensibility for future problems.

3.1 Overview of prototype environment

The Janus prototype models a small world of 100 by 100 tiles where each tile can only be occupied by a single item at a time, such as a resource\(^1\) but may contain multiple AI agents. The simulation task we represent is one of a small village populated with simple AI agents that will act with regards to self preservation only and are coached by a unified control AI. The goal of this simulation if for the controller AI to explore the map that is available to it and to gather all the natural resources that it can whilst growing and maintaining the population of villagers as best it can.

1. tree's or rocks etc.

There are a number of resources available within the world including food, wood, stone and crystals. These resources are both a goal and a prerequisite to perform any building tasks. There are a number of buildings available including a house that increases the maximum amount of villagers that the town can support as well as providing a location to produce new AI agents via breeding. A farm that is capable of growing crops over time that can be harvested. As well as a few one instance only buildings such as the stable that provides a movement speed to villagers outside of the town. A blacksmiths that improves harvesting times for resources. A church that increases the longevity of the villagers.

Villagers are designed to simulate real humans and will age and grow hungry over time as well as being able to breed and die of old age. The impacts of these effects will be explained more below.

3.2 Hierarchical Task System

![Fig. 1. Cut Tree Task](image)

We developed a system of encapsulated tasks that can be assigned to a specific villager, very similar to the idea of primitive and compound tasks in HTN planning. These tasks are entirely self contained and self managing with a villager AI simply storing if the task is complete, in progress or failed. One example of this is the move to class which constructs an A* path [16] [17] across the known grid space\(^2\) to the target location. And passes the path to the parent AI directly as set of movement instructions. In this way the movement task does not manage the movement of the villager specifically, but instead monitors the villagers position, such as distance

2. Squares that have not been viewed by any villager are obscured by a fog square and cannot be pathed across as there contents is unknown.
from target. Then when a condition is met within the movement task such as the AI failing to reach the target or succeeding in reaching it the task then marks it’s current state accordingly.

This allows for easy task combination and branching logic, for instance a chop wood task may involve moving to a tree and gathering the wood and all of this sub tasks can be combined into a parent compound task. This hierarchy of tasks is easily expandable and quick to implement for any new situations. An example of this task is shown in figure 1.

3.3 Villager desires

Villagers also manage their own survival and happiness through the use of a hunger and happiness variable that are key to task performance. Hunger plays a key motivator for all tasks that will be explained further with the unified AI controller. When a villagers hunger drops below a certain threshold they will abandon the current task and proceed to get processed food from the town hall, we opted to only allow food to be eaten after it has been placed at the town hall however this could be changed to allow AI to gather and consume it directly. If a villagers hunger drops low enough then the villager will starve to death. The other key statistic is happiness, this decreases when villagers perform tasks they don’t enjoy such as gathering resources or performing other tasks. This can be restored by allowing villagers to relax in the town by performing a special relax task that instead of lowering happiness raises it. When this drops low enough villagers will also abandon tasks however it does not cause villager death. Each villager will also age over time causing death once they reach a maximum age.

3.4 Villager Master

The villager controller is responsible for managing task priorities as well as assigning tasks to villagers. It manages a data structure containing all information on resources that have been discovered within the level as well as managing all villagers.

The main responsibility of the villager manager class is to assign tasks for villagers to perform, this is done in a multi layered behaviour tree which has 3 stages each with multiple conditions and sub layers. The stages are the emergency breeding, society goals and idle villager layers these can be seen in figure 2. I will outline their functions below after first defining how villagers are selected for tasks as either of the 2 methods used for selection are used at every stage of the behaviour tree.

3.4.1 Villager suitability

Before villagers are assigned tasks to complete basic checks are performed on the villagers to ensure they are suitable and will be able or likely to complete tasks. This involves checking the age of the villagers to ensure they are not children and that their happiness and hunger levels are high enough to have a good chance of completing a task. These numbers were found via testing the average amount of time taken to perform tasks and selecting suitable values. However these are not a guarantee and villagers can still fail (hunger/happiness drops too low). A more complete process would rely on forward running the task and evaluating whether the villager could reach the destination based on the path before the hunger or boredom thresholds cause them to abandon the task.

3.4.2 Villager selection variants

There are 2 types of search those that will find a villager that is idle for low priority tasks or a search that first looks for idle villagers but if it finds none instead finds a villager who is performing one of the low priority tasks such as gathering/exploring to perform another task. This relies on specifying a task the villager is being requested for as cancelling a gather food task to perform a food gathering task is not efficient. There is also an optional parameter that allows this important find function can have specifying the gender of the villager this is important for breeding and combat.

Fig. 2. Behaviour Tree

3.4.3 Emergency Breeding Layer

This layer is used to prevent society death from single sex societies or very low population numbers causing the society to die out. If there are less then 1 male or female in the society (regardless of age) or there are less then 4
villagers in the society then it encourages breeding even if food resources are low as starvation is easier to fix then running out of villagers.

3.4.4 Society Goals layer

The Society Goals layer operates in one of three states depending on this key metric defined for success, which in this situation is the amount of available food that villagers have stored at the town centre. It uses the amount of food in combination with the hunger of all of the villagers (information it can gather from the villagers directly) to calculate how long the society can survive knowing the rate at which villagers consume food to give it a count in seconds as to how long before starvation occurs of the last villager.

It then branches to one of the 3 states available to it expansion, improvement and crisis prevention. The expansion state is used when available time is greater then some predefined value it prioritises the improvement of the village either by breeding to increase the work force or building more buildings which are constructed based on a priority that could be specific to AIs needs but in this case is fixed. The improvement state is triggered when food is lower then the expansion tier but higher then crisis prevention and prioritises the construction of farms as well as ensuring that at least a certain number of villagers are currently assigned food collection tasks either from farms or hunting animals. And finally there is the crisis prevention state that is triggered if the food remaining time is very low in this state many villagers are instructed to gather food as quickly as possible from the available hunting locations or from farms.

3.4.5 Idle villager task layer

After the previous 2 layers have been gathered any villagers required for tasks the final layer uses a low priority search to find villagers that are still idle to assign gather or explore tasks too. These tasks are selected randomly and will assign the nearest resource of a random type or give the villager the explore task that simply causes them to walk to a nearby undiscovered tile.

After this stage any villagers that are idle and are hungry can then be sent to eat in order to reduce their hunger so that they can be assigned tasks in the future.

3.5 Villager Combat

3.5.1 Enemies types and task system

There is also a hostile version of the Janus prototype where enemies and wild animals are introduced that will attempt to attack and kill the villagers that venture to close to them. The Enemy AI functions using a separate set of Enemy tasks and are limited to two tasks a compound patrol tasks that involves patrolling between a set of nodes at certain intervals (shown as green squares connected via lines in figure 3). The other task is a compound attack task that involves moving towards any villager that ventures close enough to their central location marked in the case of the bandits as a fire (shown in blue in figure 3) and the villager is also within there line of sight range (shown in red in figure 3). There is also a high powered dragon type enemy that requires a larger group of villagers to attack and defeat it due to it’s higher health.

Fig. 3. Enemy sample overview

3.5.2 Villager reactions and retaliation

When attacked a villagers hunger meter is lowered as they eat to regain health in this simplified game world. If they are attacked whilst performing a task they will abandon the task and return to town immediately to eat to prevent their own death as a hunger of zero will trigger a villager death. The eat task is also one of the highest tier of tasks so will not be interrupted by other tasks ensuring they return as quickly as possible.

Villagers are capable of retaliating however, during the expansion tier of the Society Goals layer (defined in section 3.4.4) they are capable of collaborating to attack a enemy. The Attack action first checks for an available number of male villagers that are idle or performing any task of lower importance such as gathering or exploring. Once it has counted and made a note of all available candidates if this count is high enough they are all given the attack target defined by the Janus controller and will proceed to attack it together. They will not flee upon taking damage but will flee once their hunger is reduced suitably and leave the remainder of combat to the other villagers in the mob.

3.6 Prototype testing & results

After completing the prototype it was tested very thoroughly initially in a small 25² tile grid and then in the full 100² tile grid.

All tests were carried out on the non hostile environment the success of the villagers out of 32 runs was
4 Conclusion and Possible Improvements

The Janus prototype developed proved to be a successful lightweight and quick to implement solution to the problem space. It’s success rate was quite high with out adopting a overly cautious approach. The system is also easily extensible and can be easily customised. However after the implementation it has become apparent that while simple solutions like this are possible they will require larger and larger amounts of error checking as the state space increases without proper planning before hand as many problems faced with development could have been overcome with better system design and maps of inheritance and class interaction points.

4.1 Improvements

The addition of the combat system was developed quite late as an addition that was only really included to see the amount of time taken to implement it and expand the test world. It did not take long to include the enemies and enemy task systems however they were not structured in an optimal way due to there late addition however with suitable re-factoring they could be brought up to standard. It also helped to demonstrate the short amount of time required to implement prototype features such as the enemies however the cost then comes in due to needing to re-factor this features code. The testing carried out was not as complete or robust as was possible for the hostile mode, but could provide data on the successfulness of this technique for other genres such as RTS games or other competitive combat based games.

It would be important to test the system in other types of games using the hybrid behaviour tree to assess whether it’s quick development time and extensibility could be translated to other genres of games. Obviously other game types would require different types of hierarchical behaviour tree’s however good results could be achieved for a FPS game, with a similar approach to that of Halo 3 [13], using a specific behaviour tree corresponding to the current environment. This could include some sort of weighting system for tasks or actions that could be specified for each controller Agent in a specific environment or level, this could limit the need for more complex AI as complex behaviours could be developed through tailored weightings. This strategy may not translate to some other types such as Racing games as the tasks for these types of games are not necessarily definable in a similar way to the other genres.

It is also possible thanks to the modular nature of a behaviour tree to create AI agents like Janus that had different strategies simply by changing the priority of tasks different tactics can be developed for aggressive or passive AI as needed. And thanks to the compound tasks it is possible to abstract away certain behaviours to allow for quickly understandable and up-datable behaviour trees without the traditional problems of them being difficult to alter due to their size and complexity.

There were also problems in the prototype with frame rate for certain tasks such as the exploring task however this was caused mostly by the A* path search. However this shows that all tasks would need to ensure that they could not cause such a bottle neck as this can cause a drop in frame rate which is not good and would prove impractical for full 3D games that require more processing time for physics and other calculations. By limiting and ensuring that all tasks completed within a set amount of processing time and by spreading the planning across several frames it may be possible to simulate plans for many agents very rapidly.

4.2 Final thoughts

The test environment created for this world performed very suitably and thanks to the easily extensible and quickly changeable nature of the behaviour tree created it could be possible to use this solution for more complex games. It is possible through adaptation that this system could work for any game that utilises simple task definable objects regardless of the prerequisites of each task given a correctly defined behaviour tree.
REFERENCES


APPENDIX

Average Successful Time \(00:36:33\)

Average Failed Time \(00:07:25\)

Average success Rate \(90\%\)

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Test Successfulness</th>
<th>Time Taken (HH:MM:SS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FALSE</td>
<td>00:09:28</td>
</tr>
<tr>
<td>2</td>
<td>TRUE</td>
<td>00:35:09</td>
</tr>
<tr>
<td>3</td>
<td>TRUE</td>
<td>00:35:11</td>
</tr>
<tr>
<td>4</td>
<td>TRUE</td>
<td>00:37:19</td>
</tr>
<tr>
<td>5</td>
<td>TRUE</td>
<td>00:35:34</td>
</tr>
<tr>
<td>6</td>
<td>TRUE</td>
<td>00:34:26</td>
</tr>
<tr>
<td>7</td>
<td>TRUE</td>
<td>00:36:32</td>
</tr>
<tr>
<td>8</td>
<td>FALSE</td>
<td>00:07:25</td>
</tr>
<tr>
<td>9</td>
<td>TRUE</td>
<td>00:36:29</td>
</tr>
<tr>
<td>10</td>
<td>TRUE</td>
<td>00:33:57</td>
</tr>
<tr>
<td>11</td>
<td>TRUE</td>
<td>00:36:52</td>
</tr>
<tr>
<td>12</td>
<td>TRUE</td>
<td>00:38:36</td>
</tr>
<tr>
<td>13</td>
<td>TRUE</td>
<td>00:38:20</td>
</tr>
<tr>
<td>14</td>
<td>TRUE</td>
<td>00:36:07</td>
</tr>
<tr>
<td>15</td>
<td>TRUE</td>
<td>00:38:44</td>
</tr>
<tr>
<td>16</td>
<td>TRUE</td>
<td>00:36:42</td>
</tr>
<tr>
<td>17</td>
<td>TRUE</td>
<td>00:37:29</td>
</tr>
<tr>
<td>18</td>
<td>TRUE</td>
<td>00:36:05</td>
</tr>
<tr>
<td>19</td>
<td>TRUE</td>
<td>00:38:57</td>
</tr>
<tr>
<td>20</td>
<td>TRUE</td>
<td>00:41:36</td>
</tr>
<tr>
<td>21</td>
<td>FALSE</td>
<td>00:05:22</td>
</tr>
<tr>
<td>22</td>
<td>TRUE</td>
<td>00:36:21</td>
</tr>
<tr>
<td>23</td>
<td>TRUE</td>
<td>00:38:04</td>
</tr>
<tr>
<td>24</td>
<td>TRUE</td>
<td>00:35:55</td>
</tr>
<tr>
<td>25</td>
<td>TRUE</td>
<td>00:36:09</td>
</tr>
<tr>
<td>26</td>
<td>TRUE</td>
<td>00:36:28</td>
</tr>
<tr>
<td>27</td>
<td>TRUE</td>
<td>00:37:04</td>
</tr>
<tr>
<td>28</td>
<td>TRUE</td>
<td>00:36:32</td>
</tr>
<tr>
<td>29</td>
<td>TRUE</td>
<td>00:36:37</td>
</tr>
<tr>
<td>30</td>
<td>TRUE</td>
<td>00:34:51</td>
</tr>
<tr>
<td>31</td>
<td>TRUE</td>
<td>00:39:10</td>
</tr>
<tr>
<td>32</td>
<td>TRUE</td>
<td>00:36:11</td>
</tr>
</tbody>
</table>

TABLE 1

Average Results

TABLE 2

Testing Results