Instituto Superior Técnico

Artificial Intelligence for Games, 2019-2020

4th Project – MCTS agent for Don't Starve Together

Group #14 (MEIC-T)

José Teixeira 81937

Rafael Ribeiro 84758

Francisco Barros 85069

I – Implemented Actions

For each of the following, implemented, actions Walter first verifies a set of preconditions, listed below:

- *AddFuel*:
  - Has a flammable material and has a nearby campfire or firepit.
- *Construct*:
  - Has enough ingredients to build the respective item or structure.
- *Eat*:
  - If the *invobject* Walter is trying to eat, is in fact Food.
- *Equip*:
  - Verify that the *invobject* is of type *Item*.
- *Feed*:
  - Verify that the target we are giving *Food* to accepts it in its diet.
- *Fight*:
  - Ensure Walter has an equipped *Weapon* and the fighting target is an *NPC.*
- *HoldPosition:*
  - No checks required.
- *PickUp*:
  - Verify that the target is pickable by hand or if a *Tool* is required.
  - If a *Tool* is required, we ensure Walter has it and equips it.
  - To prevent repeated *PickUp*, GUID is removed from *PreWorldState*.
- *Unequip*:
  - No checks required, because *Unequip* action is only added to Walters' *AvailableActions* when an *Item* is equipped.
- *Wander*:
  - No checks required.

II – MCTS

II.I – Heuristic

Following Ricardo Quinteiro's approach, we decided to use a simple Heuristic based on the Light and Food values already provided on the professor's code. We felt considering equipped items does not really reflect the quality of a state, since the objective of the game is to survive. Furthermore, we did not expect Walter to live for long periods of time early on. We set our goals to ensuring he survived two or three nights and for that, Walter needs only to care about food and light, both of which can obtained using hand-only picking. However, since the game has three different times of day (day, dusk, night) we decided to extend the proposed Heuristic to distinguish day and dusk, as well. During the day, the H value is equal to 20% of the *LightValue* and 80% of the *FoodValue*. Unfortunately, we could not infer how useful this improvement was, because our agent struggles to survive the first night. In some cases, Walter 'dies' during the day, because the AI crashes, when Walter is attacked. The group did not find a way of fixing this issue since *Hineos* documentation and even the brief mentions on both Fábio's and Ricardo's reports, do not properly explain how to deal with *Events*.

II.I – Playout Depth vs. Selection Depth vs Max Iterations Per Frame

Just like in the previous section, the group decided to go with low values, choosing *selection depth* and *playout depth* to be set to 2. In this game, it is more crucial for Walter to decide quickly than it is to decide 'properly'. We are not saying that changing these values does not have an impact; however, measuring the quality of the decision is hard unless the agent was to survive dozens of days. Since the generated world is random and the availability of resources is also arbitrary, there is no real baseline comparison, and since the faster Walter can decide, the more actions per minute he can output, he increases the *odds* of survival due to the sheer number of gathered resources. Along with code optimizations the group made to the existing *PreWorldModel* and *WorldModel* classes, we are confident that the rate at which Walter is queried for a decision can be safely increased.

III – Implementation Decisions & Optimizations

The original code was very unreadable, and maintainability was difficult, this led us to replace in trains of if-else statements that compared strings with a more objected oriented approach supported by the used language, C#. This change not only increased the quality of the code from a perspective of software engineering, but also increased its effective running speed due to the massive reduction of branching done in the code. We accomplished this goal by creating a *Food.cs*, *WorldResource.cs, Buildable.cs* and *NPC.cs* files, which leverage class inheritance to define simple and common behaviors between game entities. Within this files the Singleton design pattern is also used, to avoid constantly creating new instances whenever a new entity in the game is found, e.g., all entities with *prefab* equal to 'twigs' are mapped to a single *Twig* instance that provides several information regarding the entity's capabilities. This change alone was not enough to reduce if-else statements, in fact, it would make them worse, because comparing class types is slower than comparing strings, thus, singleton dictionaries were created mapping the *prefab* to the respective *singleton* instance.

The group made big use of both Dictionaries and HashSet structures, not only on the files mentioned above, but also on the *WorldModelDST* and *PreWorldState* classes, since they provide O(1) searches and also reduced the number of $O(n^2)$ and sometimes even $O(n^3)$ collection iterations to only a handful of them spread throughout the code and only when absolutely necessary.

To reduce the number of confusing Pairs and Tuples created, some *data* classes were created, e.g., *WorldObjectData*, *FireData*, within these, some Pairs were replaced with primitive types when it came to representing positions.