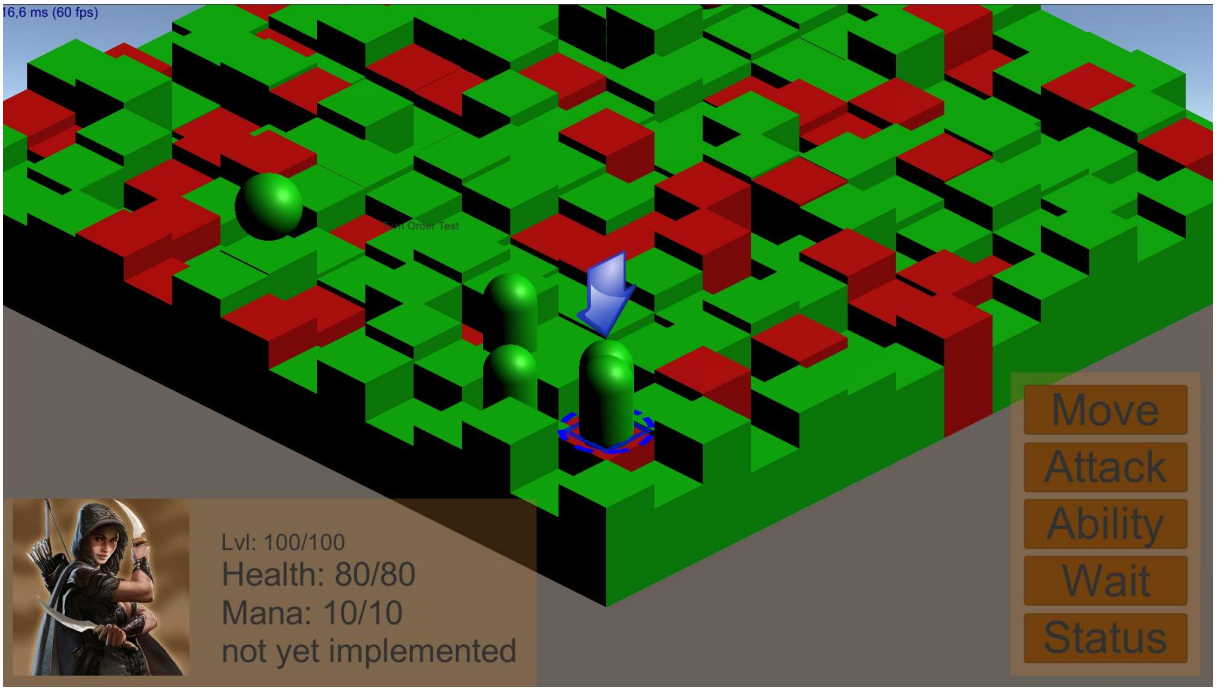
Tactics Game
AI

Bertus Jansen

GDV3 – AI

Introduction

This project was intended to become a Final Fantasy: Tactics like game, a narrative driven isometric tactical combat roleplaying game. The intent for this was to eventually build this into a full game, as such there's a decent amount of groundwork, but the game itself isn't that far yet.
The game itself is limited to a simple battle between a group of your own units, and a group of AI-controlled opponents on a semi-randomly generated field.



Iterations

Singletons

As is usually the case, I initially relied too much on the use of singletons. As a result I refactored most of the code half-way through in an attempt to reduce this. During this I also greatly reduced the amount of Monobehaviour scripts in the project. While this isn't necessarily a good thing, it did make parts of the code feel a lot cleaner than it had been before.

The AI

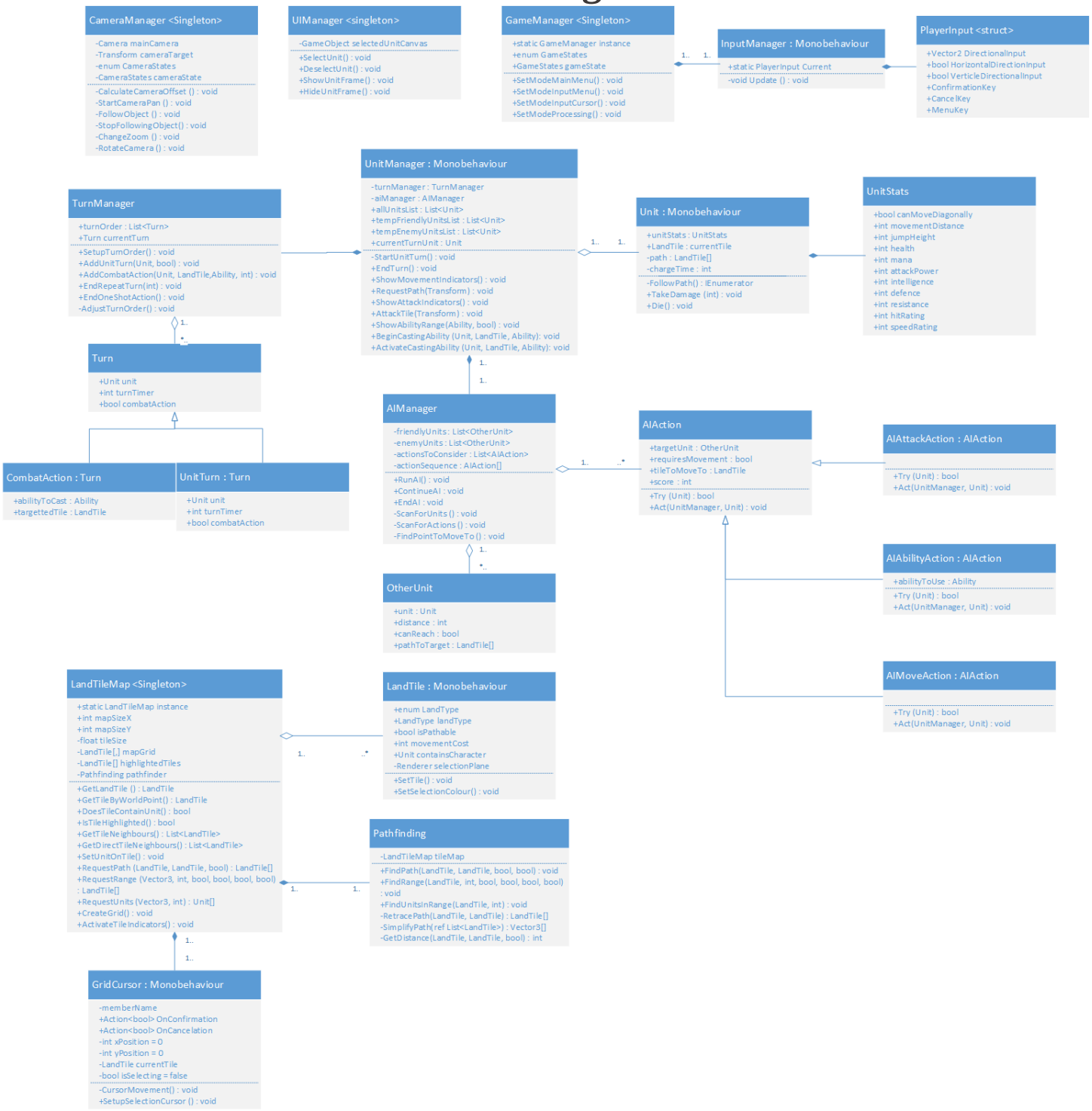
I have considered a lot of different methods for the AI, but eventually went with a Utility-AI.
Initially I was thinking about a state machine due to its simplicity. However, due to the AI having to reconsider its state and options each turn, the state machine kind of loses its purpose (maintaining a state of behavior).
A behavior tree seemed to fit a lot better, due to it starting its behavior decisions at the top of the tree each time. However, I ended up having a lot of trouble (and eventually failing) to design a tree that would properly account for all the possible options and variables that it would eventually have to deal with.
A Goal-based AI didn't really seemed to fit from the start, as the board state can change on a turn by turn basis. I do intent to eventually work some of this design pattern into the AI, but not as a base.
The Utility-AI pretty much instantly described what I was looking for. It's perfect for considering a large amount of options and choosing the best one, and seems like the one that would get the most worth out of a turn. But what interests me about it more is that it's the most 'analog' of the options. As decisions aren't a simple "yes or no", it means you can get a lot more subtle effects in there, making the AI a lot more human-like and less predictable. One downside is that you can practically keep tuning a Utility-AI to try and get it perfect, but is almost extremely difficult to actually get right.

Scoring Abilities

I had a lot of trouble considering how I would have to score abilities with unique effects for an AI. After a lot of reading I found an article on how the AI was done in the Sims, where a lot of the decision making is done by the items in the environment rather than the character itself. Rather than looking what a character wants and scanning for that, each item would 'advertise' itself. Telling the character what it could offer, and the character deciding on one based on how much it valued each one.

Similarly the movement was a fairly difficult thing to consider. Of course units could simple charge in and attack (as the AI currently largely does). But how do you make an AI understand what direction to flee in? While looking at chess I realized that step one would probably be to avoid tiles that are threatened by an opposing piece. Similarly in a tactics game, the safest tile would likely be as far away from the opponents as possible. By scoring a tile based on its distance from each enemy, and taking the lowest of the possible options. I haven't yet figured out how to get out of corners yet, though.

Class Diagram



GameManager is a singleton that handles the overall state of the game, as well as scene loading. LandTileMap is a singleton that contains all the LandTiles, and offers global access to obtain information about the board and the state of individual tiles. Pathfinding extends on this with pathfinding-related algorithms.

UnitManager is contains all the references to the units on the map. In addition, it handles all unit-related logic (including movement, attacking, and ability usage). The units themselves only hold individual data, and handle individual things such as animations, sounds or movement. AIManager adds to this by calculating AI-unit turns. TurnManager handles all turn-related logic for each unit.

The AI

The AI is based on a simplified version of the Utility-AI; meaning that it considers all its possible actions and assigns a score on each of these actions. After this it will choose an action based on which of these has the highest scores. For these scores the AI currently only really considers damage and healing, but this is intended to be expanded on in the future.

AI flow

- AI is called
- The AI scans for all units that it could potentially reach
  - First scanning for friendly units
  - Then scanning for hostile units.
- For each of those units it scans what moves it could potentially use on that unit
  - First checking for its normal weapon attack
  - Then checking for each of its offensive abilities
  - Lastly checking for each of its beneficial abilities
- The AI sorts the actions based on an assigned score (based on the damage and healing).
- The AI keeps testing/attempting actions, starting with the highest scored one.
  - Once one of these action succeeds it's locked in.
- If the action requires the unit to move into range it will create an action to do so.
- Otherwise it will calculate where to move to after the action
  - If it's a ranged/caster it will move to the least 'threatened' square
  - Otherwise it will seek to move to the lowest health enemy nearby.
  - If none are in range it will move its movement towards an enemy.
  - If all else fails it will pick a random eligible tile in range to move to.
- Once all this is decided it will lock the action and movement into a sequence and perform that.

While initially I did have a fairly decent idea of what I was going to do for the AI, once I got into it there turned out to be a whole bunch of things I hadn't considered. I think the primary issue was writing the entire thing at once before testing it. This was partly because of other issues popping up with the AI and the turn order, for example. This meant that 'testing' the AI became a case of constant errors in different lines of codes every time, without the ability to properly test if that specific issue was fixed or made worse. As a result the code has become somewhat messy, especially with a lot of added "+1's" and checks added during bug fixes. Especially things like arrays and iterating through those have become a complete mess. Once I continue on this part of the project I'll most likely scrap the entire thing and remake it from scrap; this time with a better idea of what to look out for. My first goal would probably be a system to stress test the AI's code, without having to rely on the actual in-game AI just doing its thing.

Future Additions

There were plenty of things I did look in to but didn't have time left to implement into the AI, and these are things I will most likely start adding to this project in the future.

Currently the AI just simply looks at all the options and picks the action that has the most 'value', currently meaning just damage or healing. In the future I would like to add for the AI to analyze itself and the teams as a whole, and influence its decisions based on that. For example, if it were in the lead it would 'go in for the kill', and forgo healing for the sake of dealing more damage. This would bring some life to the AI, and make it less static.
Additionally I would like to add in the ability for the AI to focus specific goals for a longer-term. For example it could concentrate on defeating a specific enemy first. If the AI were to be on the back foot, it could focus on securing a specific part of the map first and try to hold a defensive location there.
As a result the AI could eventually become a Utility-AI at its core, with some of its values decided by a behavior tree, and being steered by a GOAP system.

In the future the AI is going to have to consider a lot more complex mechanics. For example it's going to have to consider delay on spells; status effects; preferring to attack from certain angles; and spells having area of effect. In addition to this, the script has a lot of performance that could be improved on, though that's currently not the issue.

Conclusion

This project has suffered a bit from poor planning on my part. I intend to eventually expand this project over time, and hopefully get it to be a complete game at some point. This has resulted in a lot of spent time on building a good basis for the eventual game, but this had to be completely messed up and constantly band aided in an attempt to get the AI working in time. However every fixed issue introduced three more, and in the end the entire thing more or less just fell apart to the point where redoing the entire thing is the best solution.

Despite this, I did very much enjoy reading about, and working on the AI. I'm genuinely considering focusing a lot more on this in the future. It's a very interesting subject, with almost infinite room to expand and play around with. Originally I considered AI as mostly being done in state machines, which is fun and all, but not that interesting. But discovering things like the Utility-AI and Goal-Oriented AIs and the more subtle controls you have with that did peak my interest a lot more.

As for the AI itself; it's passable for a first time really trying my hand at this. A lot of the issues that cropped up mentioned earlier seem obvious in hindsight. Nevertheless I feel like I could make a far more competent AI next time.