Recommendations for extensions to the game engine

Enoch Leow 30600022
Tim Jordan 31479391

1) Add functionality to tick over other Miscellaneous Classes

Issue

There is no clear way for the game to be able to tick over miscellaneous features such as the sky class (to have the possibility to rain every 20 in game turns). This is because the game engine only ticks over ground, item and actor objects each turn. The world class does not allow class objects that are not within the game map to tick, which is the case for Sky. As a result, we ended up using the Player playTurn to tick over the sky class; however, this could cause major issues in the future in which multiple players will be implemented into the game world.

Proposed Solution

Create a new “parent class” (similar to ground, and item) that can be utilised by classes that implement in-game events which also has the tick() method that the World class will call throughout the duration of the game run time. Therefore, classes such as sky can extend from this parent class and be ticked every turn.

Advantages

1. Reduces the data accessibility Player has over in-game events such as Sky
2. Reduces the amount of coupling Player class object has
3. Easier to maintain

New Functionality

Designers could easily implement other miscellaneous classes that don’t fit ground, item or Actor classes which need to be ticked every turn. This could include Date, Time, Natural disasters (eg. earthquakes) and many more.

2) Revamp structure of GameMaps storage in World

Issue

Having multiple GameMaps in the game world is difficult to connect together (allowing players to cross from one another) since the engine currently stores them in a simple ArrayList. This means that there is no efficient way to tell which map is next to which and in what position (above, below, left or right) without explicitly stating it in some attribute (messy and must be done manually).

Proposed Solution

Alter the world to store GameMaps in an Arraylist of Arraylists. This would form a sort of “grid” where GameMaps can easily be added next to one another in any position. A class could also be created which automatically adds the functionality to the edges of each map which allow the player to move to the neighbouring game map according to

Map1	Map2	Map3
Map4	Map5	Map6
Map7	Map8	Map9

Graphic Representation of Map Storage

the Arraylist Grid positioning. An additional attribute “boolean isUnlocked” could be added to the GameMap to specify if the Map can be entered into.

Advantages

1. Much more efficient than the current manual approach
2. Allows for clear positioning of gamemaps, and better visualisation of concept
3. Easier to maintain

New Functionality

Developers can easily add new maps by positioning them in a specific location within the ArrayList of ArrayLists. Maps can also be locked and unlocked by altering the attribute “isUnlocked”. This allows the developer to let players unlock maps according to some requirement (e.g. money to unlock the new map or player level requirement).

3) Implementation of Game Modes in World

Issue

In terms of future implementations, if clients want to implement multiple game modes in their game with a variety of objectives, there is no template or class to build this idea. If for example, the client wants to create a new game mode on top of the sandbox and challenge mode, they would need to modify their original code in the game driver in order for it to be functional.

Advantages

1. Reduces the amount of responsibility game driver would have in adding a new game mode in the game world (SRP)
2. Enhances information hiding for Game Driver
3. Reduces the amount of code that needs to be changed in the original code
4. Better in terms of object-oriented design
5. Easier to maintain

Other additional possible modifications

- Implementing utility classes that are helpful in location tracking or actor tracking
    - Since there are potentially large amounts of interaction between actors (e.g. dinosaurs breeding and attacking), it would be very beneficial to add a utility class that will help actors find the location of specific types of actors in the game map
- Changing Display to have a method to read the entire line

Good things about current Game Engine

Despite experiencing some struggles while using this game engine, there are many well designed parts which make it easy to code in and also meet good design principles.

These include:

- Using parent Actor, Item & Ground classes to reduce repeated code
- The use of Capabilities to decrease the amount of dependencies, and reduce the need to use downcasting within class methods.

- The implementation of Interfaces such as Behaviour for better maintainability and reduce the cohesion between classes due to inverted dependency.
- Ground, Item and Actors all have their own AllowableActions, which meets object-oriented design principles
- Implementation of Action to reduce the amount of repeated code and provide easier maintainability
- Use of the tick function to ensure every object is synced with each of the World's turns (can be improved as seen above)