SWEN225 REC'N'PLAY REPORT

TEAM nUtjObs

Our Chap's Challenge implementation includes the functionality to record and playback saved games. This was implemented using the JSON classes and methods. These methods were used by getting the current maze state at the current time step, converting it to a JSON file and then saving it in a directory labelled after the time the level was started (as type long in UTC time format).

We implemented this feature by adding a new class called "Action Record". This class implements Saveable and stores a maze object and the time since the start of the level. Because this class implements Saveable, it can be converted to a JSON file.

We used the Javax Json parser for saving and loading the maze.

The Design Pattern that we used was a "Push Model", where the Observer is the "ReplayUtils", and the Subject is the "GUI". This model works by having the GUI send information to the ReplayUtils. This information is saved whenever something changes in the maze, and it pushes a maze object to the recording.

We also added a class called "ReplayUtils", which contains utility methods for saving the recordings. These methods do many things, such as:
->Setting up the directory to store the saved files. Each recording is stored in a sub-directory within the recordings directory. The subdirectory is named using the time it was created. This acts as an ID.
->Pushing an "ActionRecord" into the recordings/ID subdirectory.
->Playing back the recording once the level is finished. The recording is referenced by an ID.
->Resetting the recording
->Deleting the recording

The GUI utilizes some of the methods in "ReplayUtils" when the user is playing the game. This includes:
->Using "pushActionRecord()" to push a new state of the maze every time a player or enemy moves

The GUI also needs to switch between replay mode and regular game mode. Along with this, the GUI needs to perform functions such as:
->Resetting the recording from the start (using the D key)
->Skipping the recording to the end and moving to the next level (using the S key)
->Fast forwarding (using the F key)
->Pausing/Playing (using the Space bar)
->Stepping forward once at a time (using the </, key)

->Stepping back once at a time (using the >/. key)

These are implemented through a series of buttons that appear on the sidebar of the GUI. Keyboard shortcuts can also be used to navigate the replay.

The way this is updated is by using a Java Swing Timer with a delay of 10, meaning that it will update 100 times within a second. The ActionRecords are saved with their timeSinceLevelStart (as an int), and this number is rounded to the nearest 10 to determine when to update and display it. We figured that rounding this number to the nearest 5 (thus needing the Timer to run 200 times a second) was far more complicated and also seemed to lag out the program more too, as the Action Records and board would be saved and updated much more frequently than necessary.