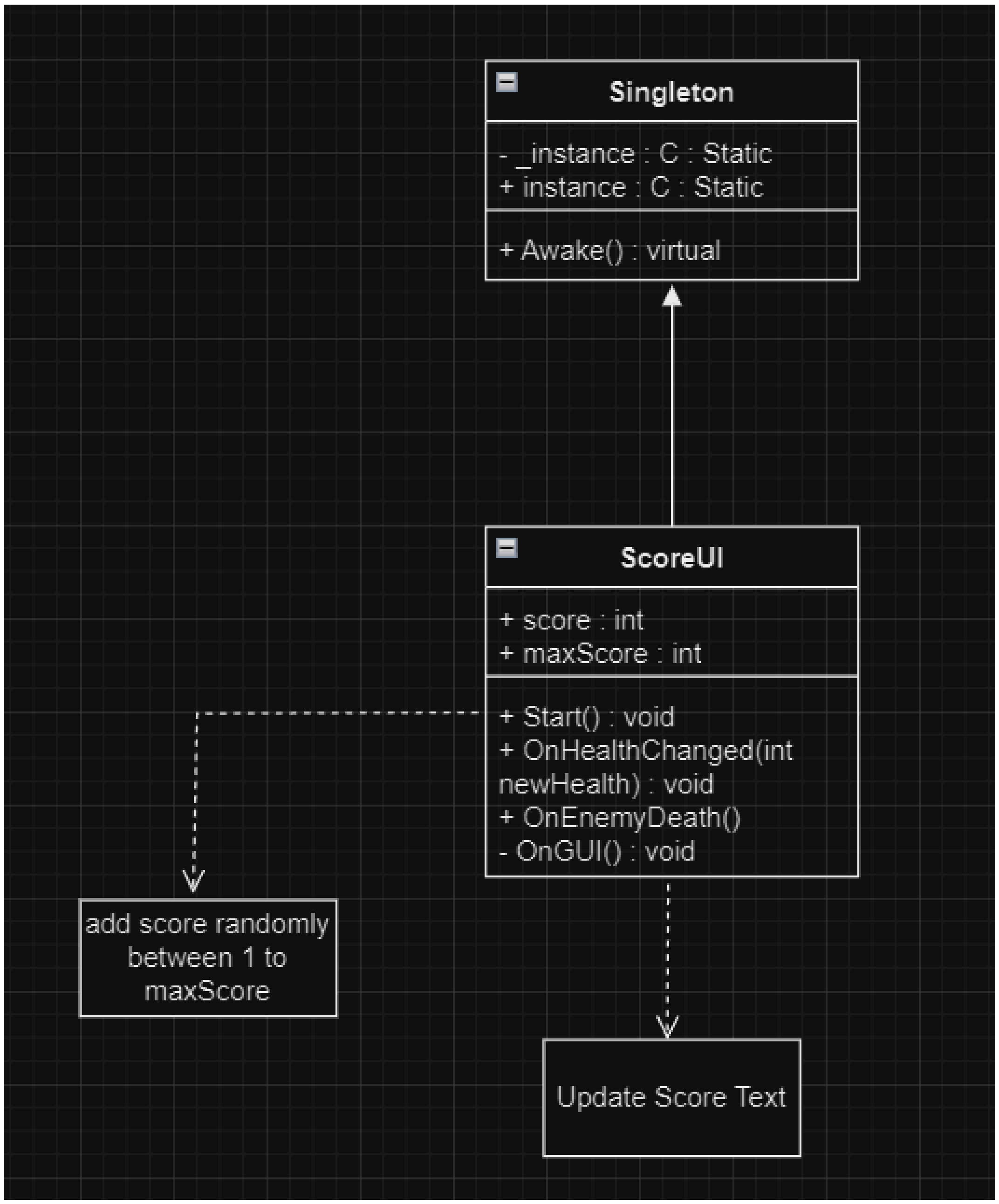
UML Diagram and explanation of implementation

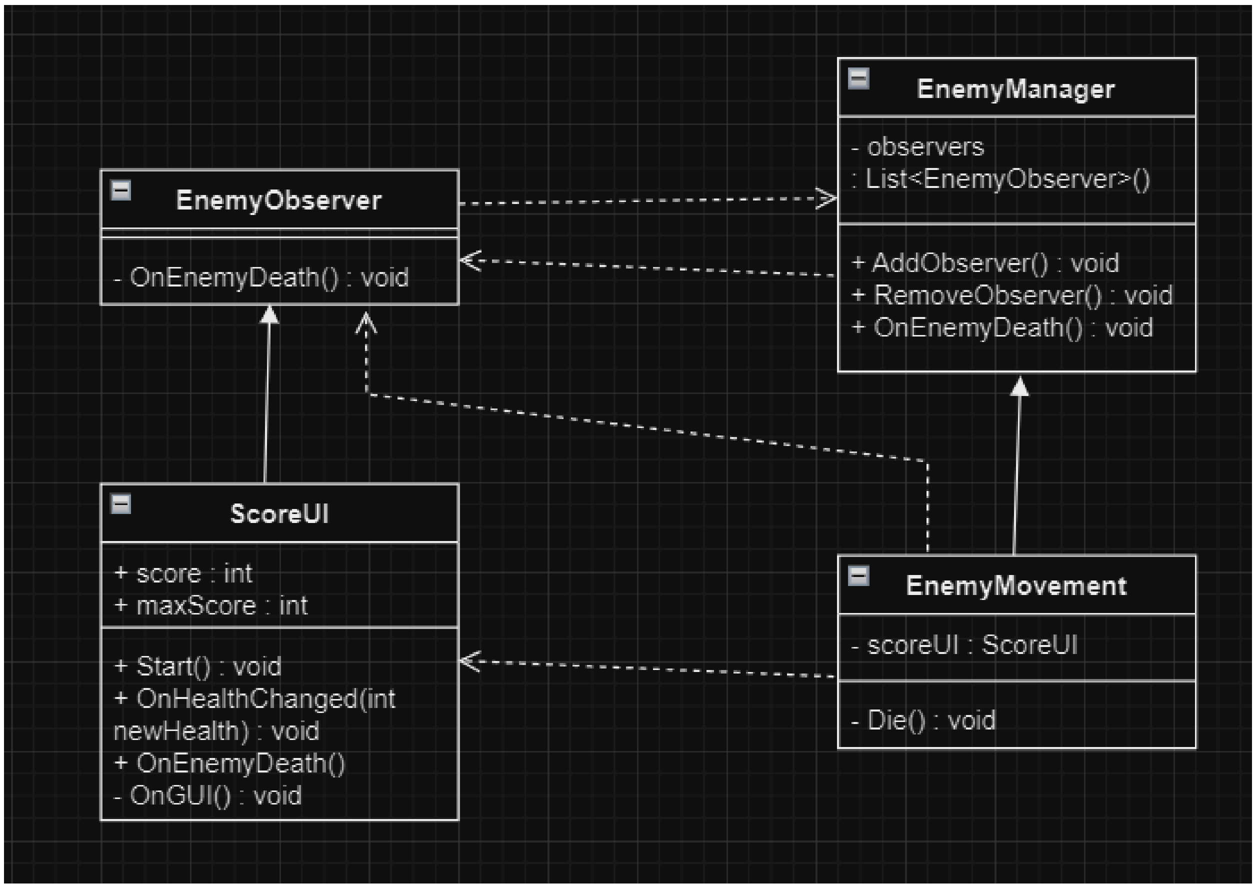
Chia Lo Chen

Singleton:



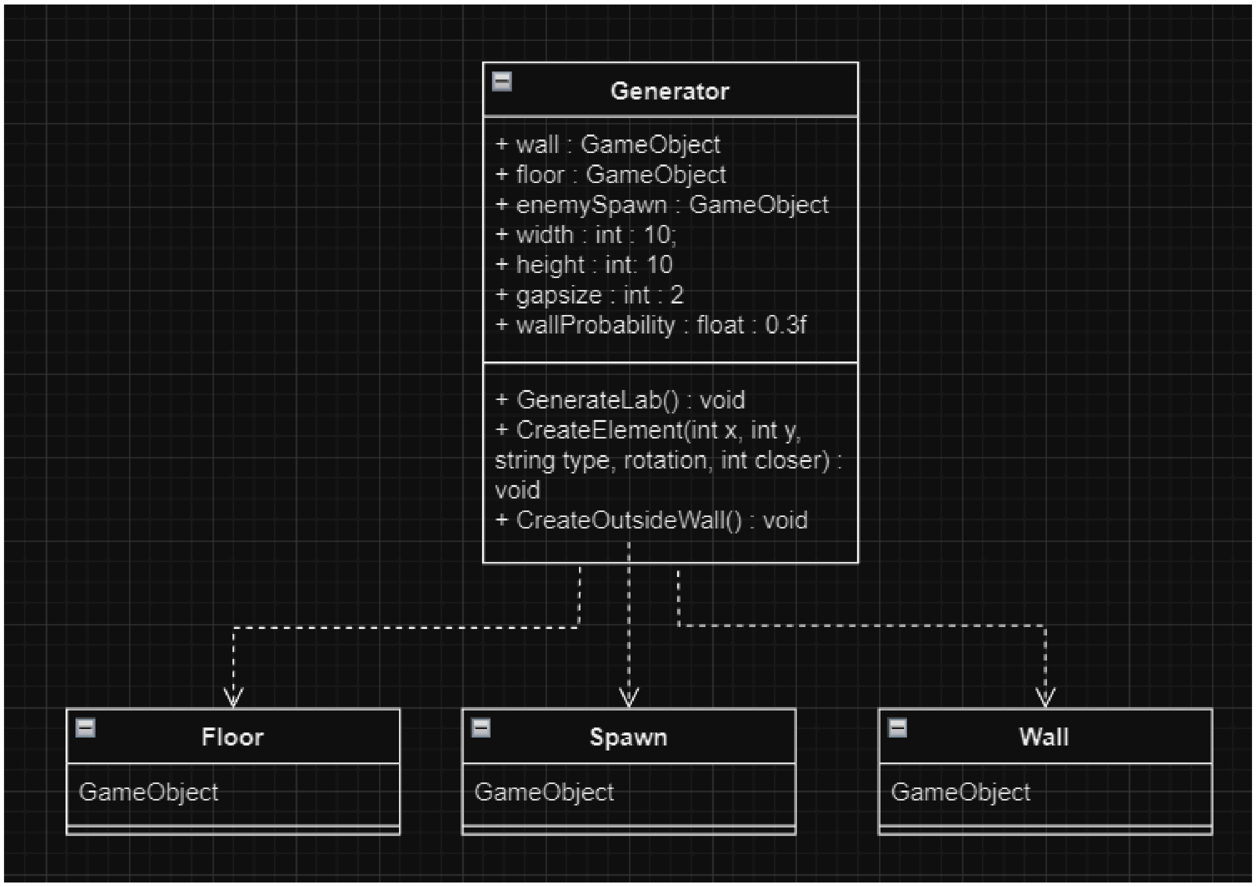
The singleton pattern being implemented is the ScoreUI system. Singleton would check if there are instances or not and create one if there isn't one, then if there's already one it would destroy the duplicate. ScoreUI would add scores randomly between 1 to maxScore, which means if there are multiple ScoreUI each of them would add a different number of scores to the scoreboard. The implemented singleton would remove other ScoreUI to avoid such a confusion

Observer:



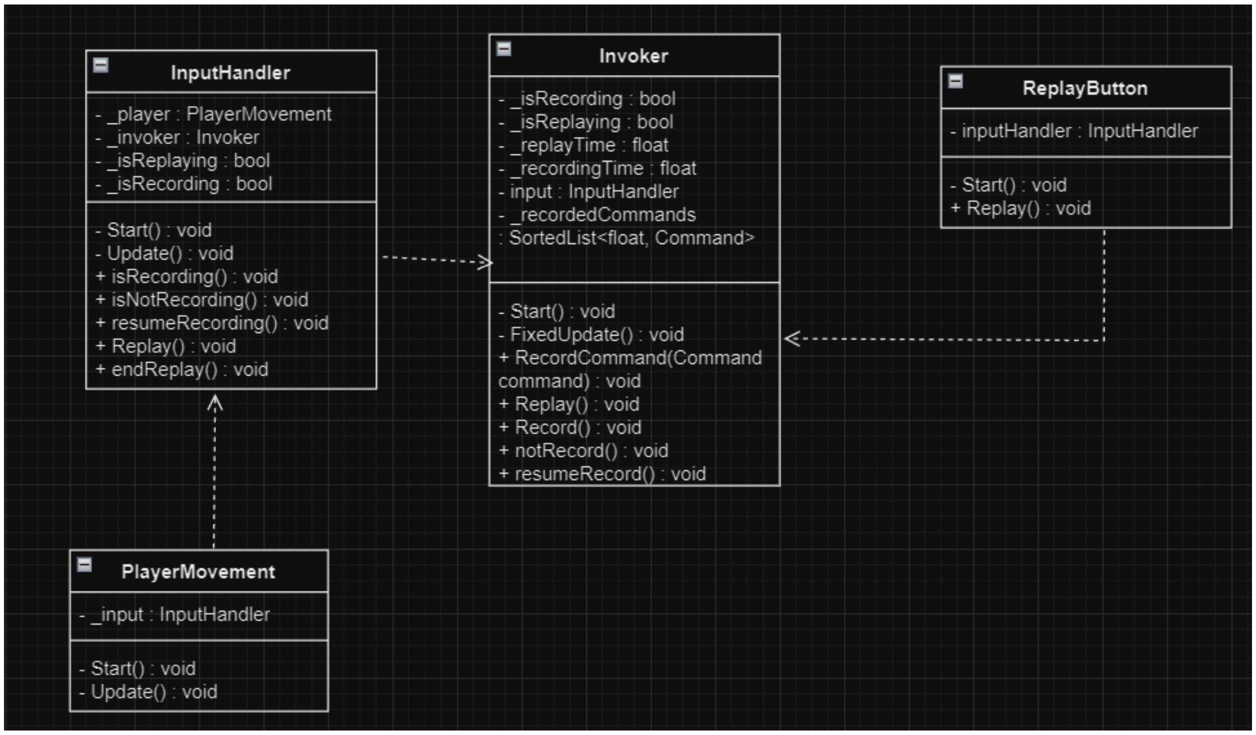
Implementation accompanied by an explanation of how it was implemented, with clear indications as to why it was implemented the way it was, and how your interactive media experience benefits from it.
The observer was implemented into the ScoreUI through subscribe to the enemyObserver while EnemyMovement would subscribe to EnemyManager. Whenever EnemyMovement state changes, it would notify the ScoreUI to add scores. It is implemented this way for the ScoreUI to accurately observe the EnemyMovement. The interactive media experience benefits from it because it would be easier to add in any new enemies in the future

Factory:



Factory pattern was implemented through using the CreateElement(). Whenever CreateElement() is called, it would check and instantiate the prefab depending on the type. It was implemented this way so whenever a new gameobject need to be instantiate it could be easily added. The interactive media scenario benefits from it because it makes the labyrinth size easily modulate.

Command:



Implementation accompanied by an explanation of how it was implemented, with clear indications as to why it was implemented the way it was, and how your interactive media experience benefits from it.

The command design pattern is implemented with a game replay system. The Invoker would record the command being called in the InputHandler and store it in a sorted list. When the player paused the menu, it would call a function within the InputHandler that would call Invoker to stop recording and the same for unpausing but continue to record instead. Then in the pause menu, if the player clicks the replay button then the ReplayButton function would call InputHandler to start replay by calling Invoker's Replay() function. When replaying, the invoker would start the replay timing and when the replay time is approximately the same as the recorded command time, then it would play the command in the first value of the list and then remove the command. It is implemented this way to accurately record and replay all the commands in the sorted list and make sure when the menu is paused, the invoker won't record any command and time as the player might be swapping keys. Our interactive media experience benefits from it because it allows players to watch their game replay.