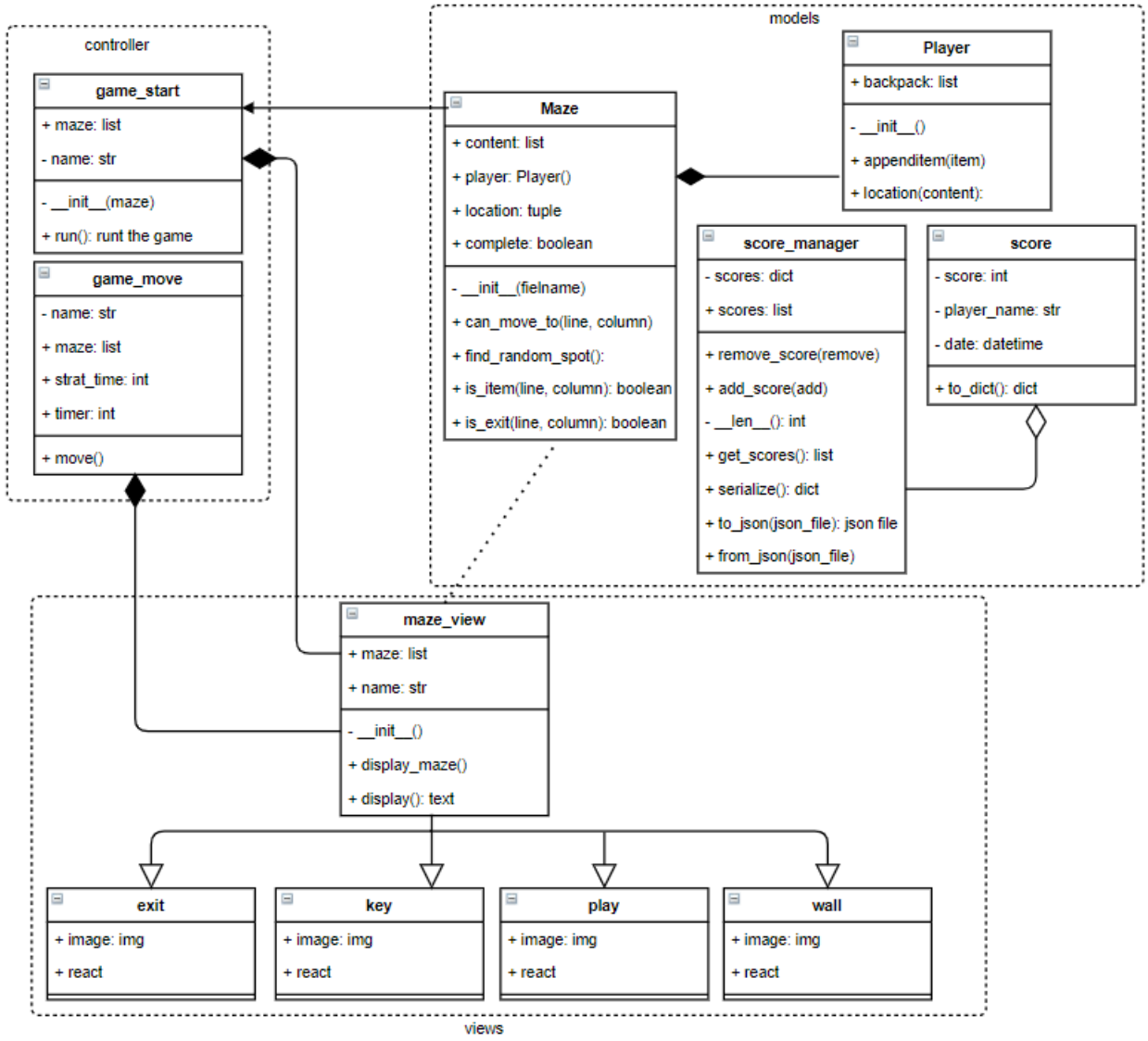
Group Project Final Report: Group 9

Roles of individual team members and their contribute

Liam was in charge of the game logic programming. He worked on creating the maze, backpage, setting up the random items in the maze. He made sure that the game can start, end, win and pick up items properly. Nicholas was in charge of UI programming and asset preparation. He worked on creating the player and controlling the player movement using a keyboard. He implemented GUI using pygame. Ishawn was in charge of general app programming and server-side/data persistence handling. He worked on creating an app and a webpage that shows the score rank. He used JSON to store the score and implement the high score model. Cheryl was incharge of project management, and report write-up. She worked on creating unit tests, a UML diagram and applying MVC templates. She designed the html code for the web page. Everyone in the group also made sure that the code documentations were in the same format.

UML diagram



Group Project Final Report: Group 9

Elaborate on the reasoning behind your code structure

We had 2 separate files that contained the maze game and the web app. Firstly, in the maze folder, we had our game_move.py and game_start.py in the controllers folder because they are the key points that make the game runs successfully. And they are the components to interact between the user and the maze game. While in the models folder, we had maze.py, player.py, score.py and score_manager.py because they hold the game and player's information. Thirdly, the views folder contains files that will present the game information to users. Fourthly, the sprites folder contains the picture that will be present in the maze game. Last but not least, the test folder contains different test files in its corresponding folder to test different areas of our code. In the web folder, we had score and score_manger in the models folder because they contained the player's information. We had a templates folder that contained a base, list and empty list html files. The base html contains the base setup code for the web page. The list html displays the score rank while the empty_list displays a note that there is no score in the score rank. We also had a test file to test our app.

How to calculate the score when a player wins

Depends on how long the player takes to reach the exit and collect all items. The shorter time a player takes, the higher score the player has. The remaining time will be the player's score. For example, if the timer is 100s and the player spends 30s, the final score will be 70.

How and why we use JSON in web API stores data

We used JSON to store all the scores. We used this approach because it can store key/value pairs. It is easier for us to store and get the correct player's name and score from the JSON dictionary. In our json file, we store the player's name, the score, the date and time they play the game.

Bonus features

We had implemented a **timer** for 100 seconds. When the timer reaches 0, the player loses if they do not reach the exit or collect all items. We implemented in our game_move.py, once the timer reaches zero, if it does not send a score to the database, it will quit the game and let the player know that they lose the game. The timer is displayed on the bottom left corner in the Pygame window. In maze_view.py, we give a timer when we call GameMove class.

Secondly, we had implemented a **backpack** feature that the player can see the item they have collected. We implemented in our maze.py, when the index of the player and the items are the same, we will add it into a backpack list. We then created keys based on the length of the backpack list and displayed it at the middle bottom of the Pygame window, next to the timer.

Thirdly, we implemented the **player name** feature where we can send a player's name and score to our web API and display it in our webpage. We will ask the player to input their name before the game starts and pass it as an argument in our game_move.py. Once we have calculated a score and if the score is not zero, we will create a Score instance. We will then add it into the score_manager list and a json file. We then can display the score with the player's name by getting from the json file.