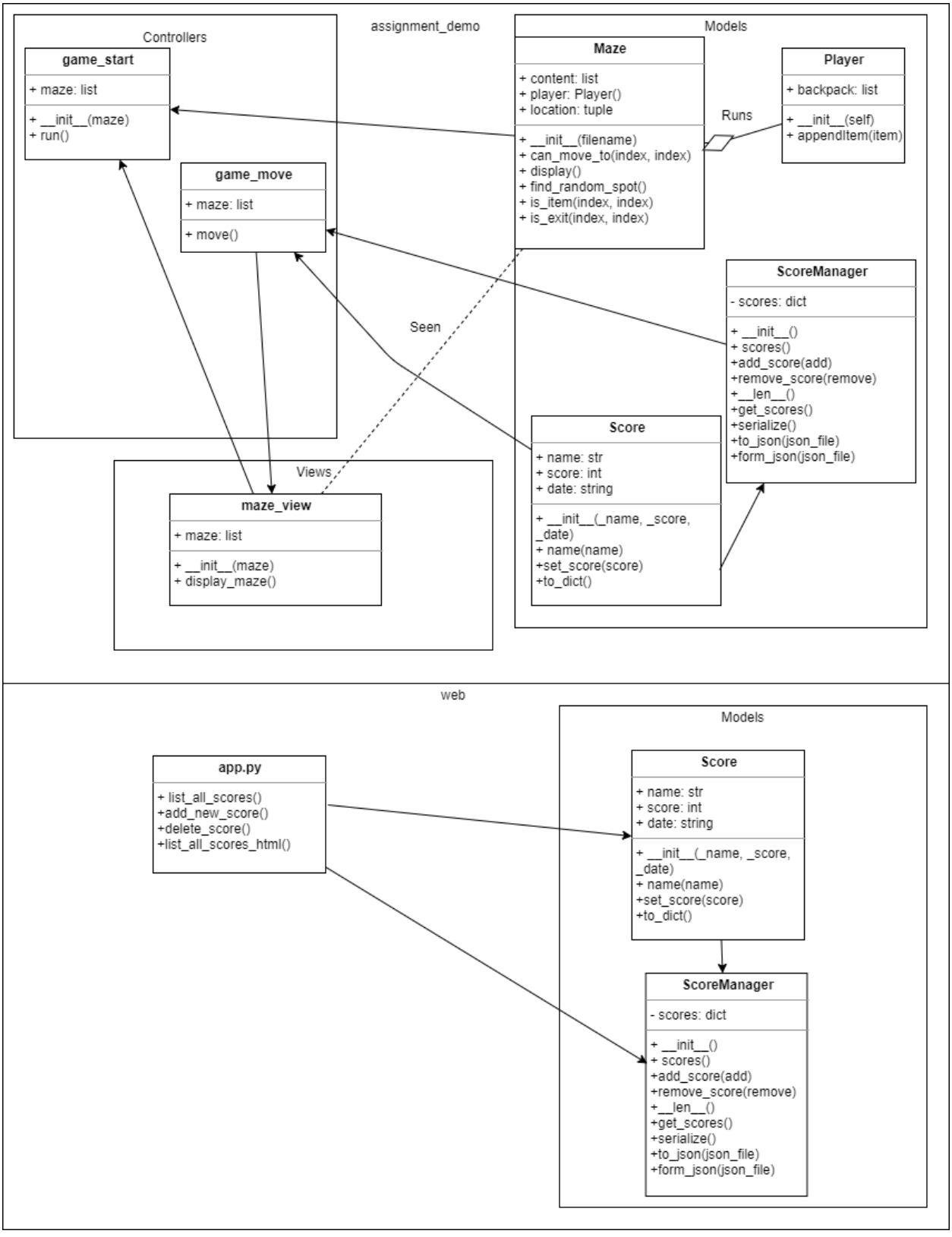
Team member roles:

- Owen Anderchek #A01211852: Main coder, pygame design, pygame functions, API and persistence coder.
- Rony Tran #A01072964: Graphics design, coding, test implementation, documentation, clean-up code/polishing.
- Brent Lee #A01207842: Coding, tests, test documentation, presentation, clean-up code/polishing.


assignment_demo
Controllers
game_start
+ maze: list
+ __init__(maze)
+ run()
game_move
+ maze: list
+ move()
Views
maze_view
+ maze: list
+ __init__(maze)
+ display_maze()
Models
Maze
+ content: list
+ player: Player()
+ location: tuple
+ __init__(filename)
+ can_move_to(index, index)
+ display()
+ find_random_spot()
+ is_item(index, index)
+ is_exit(index, index)
Runs
Player
+ backpack: list
+ __init__(self)
+ appendItem(item)
Seen
ScoreManager
- scores: dict
+ __init__()
+ scores()
+add_score(add)
+remove_score(remove)
+__len__()
+get_scores()
+serialize()
+to_json(json_file)
+form_json(json_file)
Score
+ name: str
+ score: int
+ date: string
+ __init__(_name, _score, _date)
+ name(name)
+set_score(score)
+to_dict()
web
app.py
+ list_all_scores()
+add_new_score()
+delete_score()
+list_all_scores_html()
Models
Score
+ name: str
+ score: int
+ date: string
+ __init__(_name, _score, _date)
+ name(name)
+set_score(score)
+to_dict()
ScoreManager
- scores: dict
+ __init__()
+ scores()
+add_score(add)
+remove_score(remove)
+__len__()
+get_scores()
+serialize()
+to_json(json_file)
+form_json(json_file)

Code Structure:

Our code is split between the flask app and the pygame app as needed. We did our best with the mvc in the pygame app but had issues with running the view from the controller and as such ended up running the controller from the view. Our code has a relatively straight flow, starting at main.py, calling game_start.py, which creates a maze from a text file. This maze in turn creates a player to track items with a backpack. This maze and player are then plugged into the maze_view to create the maze in pygame. Game_move.py is then called inside the view and once the player reaches the exit score.py and score_manager.py are called to create and export the score to a json file in the main project folder. This json file is then unpacked by app.py when it calls score_manager.py in the web folder, plugging it into the website to be displayed. The reason we have two score.py's and score_manager.py's is because there were issues with importing and this was the best solution we had to come up with.

Score Calculation:

Score is calculated by taking the difference of a start time and an end time created with the datetime module. The datetime minute is multiplied by 60 for both the start and end times and then the seconds from the time it was created are added together, effectively making an integer out of total seconds. As minute can be zero at either point that needs to be considered. If either start time or end time has a zero for the minute, it is then replaced by a 60 for the equation.

Web API Storage:

Our web API stores information in a json file that is in the main project folder. We chose json because it was the more straightforward of the three main options, SQLLITE requiring more models to control it and csv requiring different iteration methods. The scores are stored in the same method as shown in class, a scores key with a list of every individual score as the value. The json file is in the main folder as that is the easiest place for both apps to draw from it, allowing the maze app to find the scores and add a new one and then also allowing the flask app to call the scores and display them.

Bonus Features:

- Timer: Creates a timer counting down from 100 in the bottom left of the pygame maze by using clock.tick(60). The ticks are divided by 600 without remainder and subtracted from 100 to get the timer working, otherwise it would run too fast.
- Backpack: Displays a backpack on the border of the pygame maze screen. First creates and renders the letters backpack on the screen and renders a red cube for every key picked up in the game to the right of the backpack word.
- Player Name: The main.py file asked for input of a name that is then passed through the different classes until it is used for the player name when creating a score in game_move.py