Alan Macabuhay
Final Project Design/Reflection

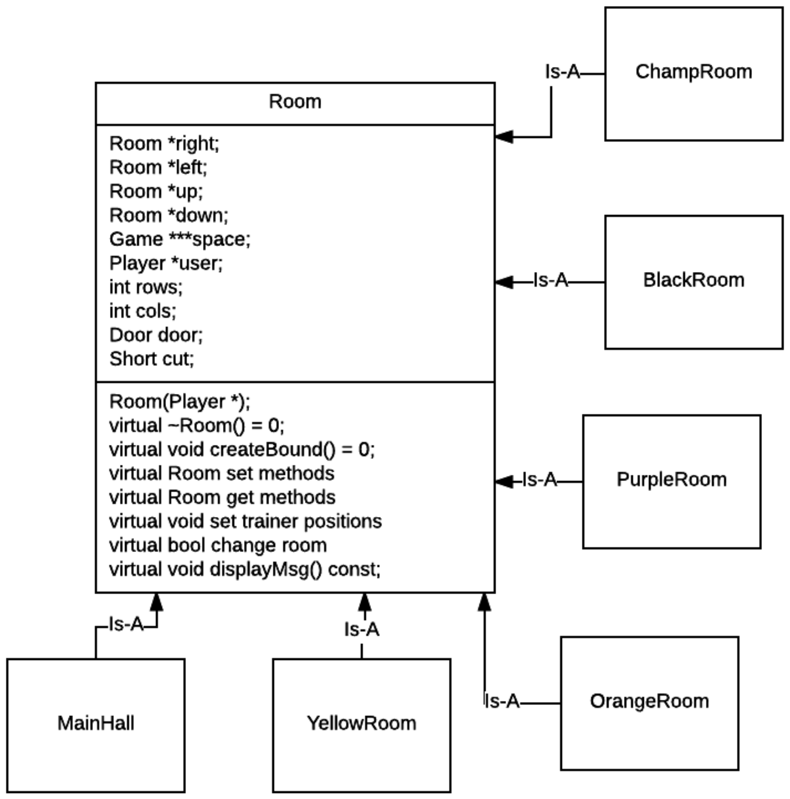
This is the final project. It is a text-based game. It uses inheritance, polymorphism, and pointers. For my project, I decided to go with a role-playing fighting game called Karate Master. The premise of the game is based off of the Nintendo game Pokemon where the player travels around collecting badges to become a Pokemon master. In Karate Master, the player will traverse a dojo and collect belts from masters in different rooms. The player must go into each room and fight trainers and masters. Defeating a master will earn the player 1 of 4 colored belts: yellow, orange, purple, and black. By collecting all the belts, the player will get an opportunity to challenge the champion. To win the game, the player must beat the champion. The fighting system is simply based off of dice rolls. It is not complicated and it was made to be as simple as possible.

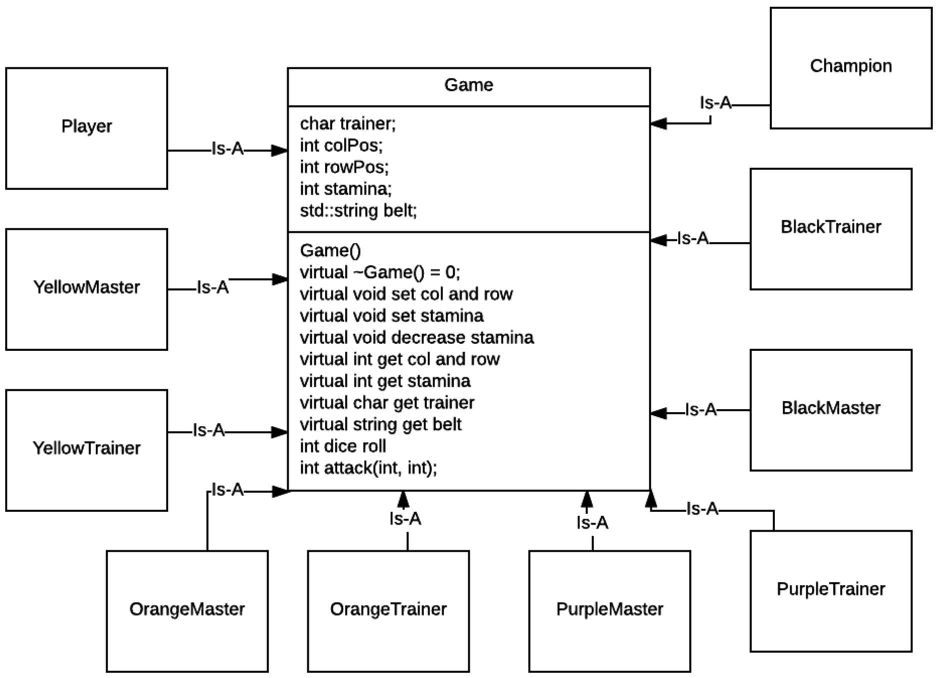
Design:
Karate Master has two abstract classes, Room and Game. I will discuss the Room class first.

Room class builds the rooms of each master and the main hall that connects the rooms. Room class also handles the player's movements and interactions. The classes that inherit from Room are Main Hall, Yellow Room, Orange Room, Purple Room, Black Room, and Champ Room. Main Hall connects the Yellow, Orange, Purple, and Black rooms together. Yellow is to the left, Orange to the right, Purple to the top, and Black to the bottom. The Champ room is connected to the Black room. Once the player gets into the Champ room they cannot go back as the game will end after the interaction with the champion.

The Game class handles the data for the player and the non-player characters. Game sets the position of the players and non-player characters in the different rooms. It also handles the dice rolling. The classes that inherit from Game are Yellow trainer, Orange trainer, Purple trainer, Black trainer, Yellow master, Orange master, Purple master, Black master, Champion, and Player.

The fighting for this game, as mentioned earlier, is just dice rolling. It's very simple and not like project 3 and 4 at all. The masters and trainers, except for having different stamina, have no other special attributes. For the interactions with the trainers, the game rolls a dice and the lower of the roll will take a set damage. For master interactions, the game rolls a dice for the player's and master's attack damage then rolls another set of dice for each, the lower of the second roll will take the result of the first roll as damage. Each master will have different number sided dice.


Room
Room *right;
Room *left;
Room *up;
Room *down;
Game ***space;
Player *user;
int rows;
int cols;
Door door;
Short cut;
Room(Player *);
virtual ~Room() = 0;
virtual void createBound() = 0;
virtual Room set methods
virtual Room get methods
virtual void set trainer positions
virtual bool change room
virtual void displayMsg() const;
ChampRoom
BlackRoom
PurpleRoom
MainHall
YellowRoom
OrangeRoom
Is-A


Game
char trainer;
int colPos;
int rowPos;
int stamina;
std::string belt;
Game()
virtual ~Game() = 0;
virtual void set col and row
virtual void set stamina
virtual void decrease stamina
virtual int get col and row
virtual int get stamina
virtual char get trainer
virtual string get belt
int dice roll
int attack(int, int);
Player
YellowMaster
YellowTrainer
Champion
BlackTrainer
BlackMaster
OrangeMaster
OrangeTrainer
PurpleMaster
PurpleTrainer
Is-A

Testing Plan:
The movement and interactions of the game is what I will concentrate the testing on. The movement is similar to the Langton's Ant project we did early in the quarter. The difference is that the player will control the movements. The controls of the game are: w = up, s = down, d = right, a = left, and e = interaction. I'll be using switch statements to handle the movements and interactions. Entering a movement input will change the player's coordinates on the game board.

Starting player coordinates: (18, 10)

Test case	Input value	Expected outcome	Observed outcome
Moving upwards from (18, 10)	w	Coordinates = (17, 10)	Results as expected. Coordinates = (17, 10)
Moving upwards from (17, 10)	w	Coordinates = (17, 10)	Player didn't move. Coordinates = (17, 10)
Moving down from (17, 10)	s	Coordinates = (18, 10)	Player didn't move. Coordinates = (17, 10)
Moving down from (17, 10)	s	Coordinates = (18, 10)	Player didn't move. Coordinates = (17, 10)
Moving up from (18, 10)	w	Coordinates = (17, 10)	Results as expected
Moving up from (17, 10)	w	Coordinates = (16, 10)	Results as expected
Moving left from (16, 10)	a	Coordinates = (16, 9)	Results as expected
Moving right from (16, 9)	d	Coordinates = (16, 10)	Results as expected
Interacting with black trainer in Main Hall	e	Trainer gives white belt to player	Results as expected

Reflection:
I wish I had more time to work on this project. I really wanted to flesh everything out and make it outstanding, but I had to settle for making it simple. I had a lot of trouble with the player's movements. Outcomes were not going as expected. Player could move once in any direction, but it did not move after that regardless of which direction I chose. It wasn't until I looked at one of the sample finals that I saw what the issue was. I wasn't updating the player's row or column coordinates after the player moved. Once the player's position was correctly updated, the results of moving became as expected.

Another thing I picked up from the sample project was creating boundaries by assigning the Door class to a space by reference. I was originally allocating too many times by creating new Doors for the spaces that needed it, but this made managing memory leaks difficult. Each room now has different designs that create a pseudo-maze that the player must traverse. This actually helped with interactions as well because I was able to control the direction the player is facing when interacting with trainers and masters. This made it easier for me as I did not have to write code for the different directions the player could be facing when doing interactions.

I also had trouble with segmentation faults. I could not figure out why until I accidentally interacted with a wall without a border. To remedy this issue, I adjusted each room so that the player will not have a chance to be next to a wall without a boundary.

Overall the final was pretty fun to work on. I just wish we had more time to really work on it.