Project Title: Street Fighter

Project Members: Winston Yang, Cheng Anze

Final presentation link: [CEP Capstone Project Streetfighter Presentation](https://drive.google.com/file/d/1Vid9FZ29tUmqe214t3711fhsv-B1OwrS/view?usp=drivesdk)

# 

# Project Introduction

Our project is a pygame rendition of the arcade classic, “Street Fighter” and we hoped to achieve a similar level of gameplay and game visuals as the original. We wanted to give players an authentic experience of the game and allow them to play against artificial intelligence that takes the form of a bot that they have to fight against in the game. This game was made to challenge players’ skills in fighting games and to give them an alternative to always fighting against a human player. Street Fighter is a game first released in August of 1987. It was a real hit; people were playing this game in every arcade there was. However, one thing stood out - the game never did increase in difficulty, as the opponent that was controlled by the computer in Singleplayer mode did not utilise reinforcement learning. Hence, it was not possible to fully enjoy the game when you could only play by yourself, and did not have another human opponent to play against. Thus, the idea of using an RL model to play against the human player was conceived. The fun of having an AI trained by reinforcement learning is that it is able to learn from what you do and use it against you. This makes the game much more interesting and appealing to the players.

# Machine Learning Algorithm

The Machine Learning algorithm that we wanted to use was Deep Q-Learning. Q-Learning is a reinforcement learning algorithm that tells an agent (the opponent) what action to take, under different circumstances. It involves an agent (the entity that carries out the action), a set of states (S), and a set of actions (A) per state. The agent changes from state to state, and executing an action in a certain state gives a certain “reward”, which is essentially a score. The agent tries to obtain the highest score possible by looking at the maximum rewards attainable in the future from its current state. With a stochastic problem such as ours, the learning rate of the machine learning algorithm would have to be lower such that the newly acquired information does not completely override old information. The data from this algorithm is stored within a table, called the Q-Table.

# Dataset and Features

In reinforcement learning, there is no fixed dataset for the AI to learn from. Rather, it creates the data by itself to learn from by playing against itself. The AI will choose an action, and that very action will affect the observation which will then affect the reward. There will be many episodes of training and they will all serve as data for the AI to learn from.

# Development

We wanted to create a custom gym environment in OpenAI to facilitate machine learning, but we did not manage to do it.

# Application Deep Dive

The application we created is essentially our rendition of “Street Fighter”. Even though it may not have the complexity and the variety of the actual street fighter game, it has the most essential features that enables players to play the game. Since it is mainly an arcade game that can only be played in traditional arcade machines, we wanted to create an application that allowed people to play street fighter using their computers. The application is very intuitive and features a simplistic menu that features two buttons, “Instructions” and “Play Game”. When you click on the play button, it allows you to choose from a total of four unique maps to play in. After selecting the map, it will then show the game screen. To move the player, you simply use the keys WASD to move the player. W to jump, D to move right, A to move left and S to crouch. Beyond the movements, the player can do four other basic attacks that are the jab (J), the heavy punch (K), the high kick (M) and the normal kick (N). These movements can also be modulated by pressing the forward key (D) at the same time as pressing the attack keys, which gives the player more damage. Each move in the game is assigned a certain damage value, as well as a cooldown value. Essentially, moves with higher cooldown values can be used only after waiting a longer period of time, and in order to balance the game, attacks with higher damage values have larger cooldown values.

Furthermore, there are also two other special moves that the player can do, which are the hadouken and the shoryuken. Players can fire off a projectile (hadoken) to deal damage to the enemy player. Finally, the shoryuken, which is a combo move, requires players to press both the W and I keys to execute this move. Combo moves such as these are parallel to the actual street fighter games that feature many different types of combo moves. However, the fact that we have such a huge variety of moves is in fact one of the main limitations of our project, as our AI would have too many actions to execute.

The playable character in the game is a pygame.sprite. It is assigned positional values (self.rect.x, self.rect.y) to determine its position on the screen, a health integer value (self.health set to 500), directional velocities (self.speedx, self.speedy), a few bool values to detect if the character is in a certain state (self.hit2, self.jump, self.crouch, self.hadoukenstate), an image and a mask (self.image, self.mask), with the mask obtained from the image, a cooldown value (self.cooldown), as well as a list that contains all the player’s actions (self.lastaction). The actions (which are essentially the states of the character as explained below) are appended to the list every time the game detects that the current state is not the same as the previous state.



A new hadouken ball is created every time the player uses the hadouken move. It is a separate sprite that spawns in front of the character and moves in the direction that the player is facing (i.e. to the right). Mask collision is used to detect if the hadouken ball collides with the opponent player. And if that is the case, then the hadouken ball sprite is killed.

The characters have several states. The states are as follows: idle, moving, jumping, jab, forward jab, heavy punch, forward heavy punch, kick, forward kick, high kick, forward high kick, shoryuken, hadouken and hit. When the user does not press any key, the character is by default in the idle state. There are 4 png files that are cycled through when the player is in the idle state, which creates the animation effect of the player bobbing up and down. A similar logic is employed with other moves - png files are cycled through to create the animation effect. This is coupled with the character being displaced to achieve certain effects, such as the y value of the player decreasing when executing a jump, or the x value increasing when the player walks forwards. The only state that cannot be achieved by pressing certain keys is the “hit” state, as that state can only be achieved if the opponent hits your character.

We used mask collision to detect collision between players. Should the player be in the state of an attack and in collision with the other player, it will deal damage to the other player and they will be hit. However, the abilities cannot be used infinitely and have cooldowns. In order to facilitate the “cooldown” effect, when the move is initiated, the current ticks are obtained, and for a period after the ticks were obtained, pressing the same key combination would have no effect. This is achieved by constantly obtaining the current ticks every frame in the game. After the cooldown period is over, the player is able to use the move again.

Health bars are also present at the very top of the window, giving the player a graphical representation of their character’s health and the opponent’s health. In between the two health bars is the pause button, which the player can use to take a break from the game. Once clicking the “resume” button on the pause menu, the game resumes from where it left off. If one of the characters’ health goes down to zero, a screen pops up, displaying that the other character has won, and the player can simply quit to menu in order to play another round of street fighter.

In summary, the character that we added to the game simulates the character “Ken” from the original street fighter, with his set of moves and animations. We believe that we have managed to effectively execute this by using pygame.

# Conclusion

Ultimately, we are proud of the functional and playable street fighter game that half resembles that of the actual game.