Working Title: *Mortal (Q)ombat*

CS 238 Project Proposal

Anthony Weng, Aviad Lengo

Due: January 20, 2023 - 5:00 PM PST

1 Assignment Requirements

1.1 Problem Overview

Brawlers are a sub-class of video games in which a player pilots a character (typically called a "fighter") against another fighter, controlled by another human player or a pre-programmed AI. Classic examples include titles such as *Mortal Kombat* (the namesake of this project), *Tekken*, and *Sega vs. Capcom*.

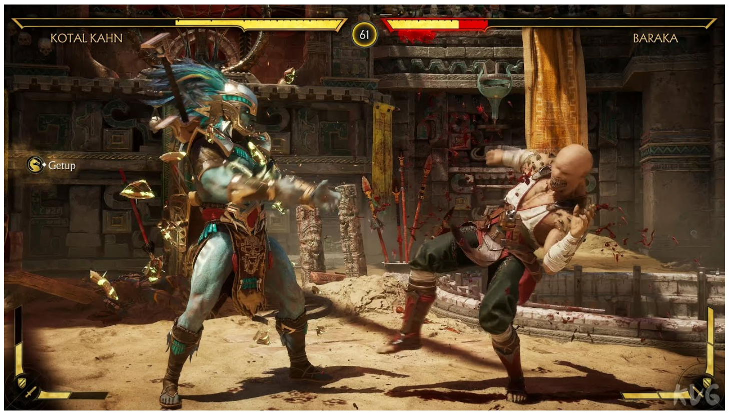

Figure 1: An example of the gameplay from the latest installment of the *Mortal Kombat* franchise, *Mortal Kombat 11*.

These games are differentiated by their character rosters, the types of actions character can take, and other gameplay elements like a story or campaign mode, music, etc. However, most if not all brawlers share this feature: the winner of a given fight is the individual who pilots their fighter with more "skill." "Skill" in brawlers can be described as a combination of *proactivity* and *reactivity*: the former being the able to predict what the opponent's fighter is going to do and plan the actions of one's own fighter accordingly, and the latter being the ability to make dynamic adjustments to regain control of the fight when an opponent's behavior deviates from one's prediction.

For our CS 238 final project, we will attempt to solve the following problem (or more generally, to accomplish the following task): to utilize reinforcement learning to program an AI agent who can play a simplified brawler game with a human-comparable level of skill.

1.2 Justification as a Decision-Making Problem

As the individual piloting the gameplay agent, there is typically one central objective: to defeat the opponent's character by lowering their health (measured as "hitpoints," or HP) to zero. However, in the course of doing so, a player has many choices. At each distinct timestep of the fight, players can select amongst a variety of actions for their character. These actions include moving their character, initiating attacks, attempting to block, etc. Many of these actions entail sub-decisions as well: for movement, which direction to move one's character; for attacks, which attack to attempt; for blocking, whether to block in a high or low position, etc.

In summary, a brawler-type game can be formalized as a decision-making problem as follows: a player piloting a gameplay agent must decide at each unique timestep what action (amongst a roster of different action types, and their respective sub-types) will be optimal in progressing toward the ultimate objective: reducing the opponent's HP to 0, thereby winning the fight and the game.

1.3 Sources of Uncertainty

For a human player of a brawler game, there are two principal sources of uncertainty:

1. The first source is uncertainty regarding what the opponent's character is going to do in the present and following timestep(s). Fundamentally, this type of uncertainty is a prediction problem.

2. The second source is uncertainty regarding what is the optimal planned action and/or reaction, given either a prediction of the opponent's action or an observed deviation from this prediction, for the player's character.

Chief to this project will be defining the gameplay environment such that these sources of uncertainty are present for our AI agent during their learning of how to play the game.

2 Additional Elements

2.1 Requested Feedback and Areas of Assistance

One of the primary areas we would like to request feedback/thoughts on is whether we should employ an existing game and RL development environment (e.g., Gymnasium) or attempt to code these elements from scratch. If we were to code up a custom game and RL environment...

- Creating the game interface, an AI agent class, and RL environment would require a significant time investment. Resources/tutorials for doing the first task (game interface creation) exist (see (1) in 'Intended Resources') and could be adapted, but the latter two elements would be a non-trivial task.

- A lack of pre-existing AI agents would require more investment in identifying a means to train the RL agent effectively as well as identifying benchmarks for performance (e.g., cannot say something like, "Our *Street Fighter* agent is able to be the expert-level AI 80% of the time!").

- We would be able to customize the state and reward spaces more freely, being able to program in behaviors like perfect prediction (e.g., opponent's current action is observed), pure reaction (e.g. opponent's past action(s)) are observed), and various intermediaries. Comparing the agents trained under these different conditions could be analytically interesting.

- We would be able to train the RL agent by having it play/fight itself, a popular paradigm for RL today which is especially interesting to us!

Leveraging an existing game and RL development environment basically offers the inverse pros/cons (e.g., less time investment in set-up, existing training partners and performance benchmarks, potential limitations in defining state and reward spaces and permitting the RL agent to train against itself). We would love to hear a course staff member's thoughts on any additional advantages/disadvantages and resources we may be overlooking as well as recommendations for what is more appropriate for the scope of this class and final project!

2.2 Tentative Project Tasks and Timeline

Note: As of now, our team has a slight preference for developing a game and RL environment from scratch. The following projected tasks will be adjusted if we opt to use an existing game and RL environment.

1. By Week 5-6 (Quiz 2): program the game environment, create an AI agent class, and sketch a plan for the intended experiments and analysis for the RL portion of the final project.

2. By Week 8 (Project status update): program the RL environment and training elements (e.g., state, reward, etc.), begin RL agent training and performance evaluation, submit project status update and begin writing the final report.

3. By Week 10 (Final project paper/video): conclude RL experiments, finish and submit final report.

2.3 Potential Resources

1. A tutorial for developing a custom brawler game using `pygame`.

2. Street Fighter AI Development using Python's `gym-retro` and the A2C model.

3. An example of an RL agent learning its optimal policy via playing itself in the context of a different game, *Stratego*.