Fighting Game AI agent with Genetic Combo Finder and Reinforcement Learning

Skyler DeCoteau
New York University
New York, USA
srd6617@nyu.edu

Yinchu Zhao
New York University
New York, USA
yz6615@nyu.edu

Wilson Li
New York University
New York, USA
tl2894@nyu.edu

***Abstract*—One-on-one fighting games strike a balance between turned-based board game and real-time simulation game in terms of AI implementation due to its strategic and reactionary nature. Reinforcement learning has been the mainstream for such research and function as a benchmark for more customized implementations. However the lack of combos hinder its performance as a human-like player and lower the performance for AI agents as well. In this proposal, using FightingICE as testing environment, we provide a genetic combo finder to discover and generate long sequences of actions not provided by the base game. Then we utilize the output of the combo finder to expand action input space and implement an improved version of reinforcement learning to master combos in the learning process. Experiments are conducted to corroborate such claims.**

I. Introduction

One of the essential elements in game design is to define a predetermined set of rules that limit the player's actions and guide them to achieve the goal. The game also proceeds the game states under the limit of the rules and player's interactions. In video games, many rules might work simultaneously, and the appearance bugs and design flaws become unavoidable. It is a time-consuming task to design the rules to include all different situations of the game. In video games that simulate physics, it is impossible to predict all states of the game. Sometimes these unpredictable behaviors can surprise the designers and players, like the "Goat Simulator" [1]. However, they may also ruin the original design of the games. Especially the multiplayer games that players compete with each other, such as fighting games.

Fighting games have been widespread since the 1990s with the release of "Street Fighter" [2], "Mortal Kombat" [3], "Tekken" [4], and "Super Smash Bros" [5]. Fighting game tournaments also appear in the fighting games community. The Evolution Championship Series (Evo) is the largest annual event in the world's fighting game community and the largest LAN event. It had been held every year since 1996 until this year [6]. In 2019, the Evo channel on twitch had more than 4.9 million hours watched.

In a fighting game, two fighters are positioned in the environment and fight until one of them is defeated. Players capabilities to execute offensive or defensive actions accordingly to opponents in a quick and precise manner is heavily tested. Offensive actions usually include punch, kick, magic or special actions while defensive side utilizes block or counter. All are accompanied with movements like dash or jump in a 2D or 3D environment with input directions. Another important concept from fighting games is the utilization of combos. Whenever a player strikes a blow to another, the player taking the hit enters a recovery animation and is unable to take actions. Should the attacking player perform another attack that is fast enough to hit an opponent before he is able to recover, this sequence of attacks becomes a combo. Combos allows actions in different states to be chained together, and adds an unique extra layer of flexibility to the game flow.

A fighting game's strategic and reactionary nature makes it a perfect test subject for AI agents implementation. Firstly, it's reactionary in a sense that players are given less time to plan future moves compared to traditional board game like DeepBlue or AlphaGo. Yet,it's also strategic in a sense that the complexity of controllable actions and the span of a match is considerably smaller than traditional MMO games like DOTA since no multiplayer communications need to be considered. A Java-written 1v1 fighting game environment called "FightingICE" provides a prefect test bed for our proposal and also functions as the main platform for the Fighting Game AI Competition. [7]

This platform can support Java and python programming to develop several AIs with flexibility: the developer can create an AI using several intelligent algorithms known to the community and initiate multiple testing. Unique attacks, combos, and movements are able to be customized depending on the algorithms implemented. The following Figure 1 showcases the general architecture for FightICE platform.

II. Related Work

A. FightingICE

Organized by Intelligent Computer Entertainment Lab. (ICE Lab.), Ritsumeikan University, Competition on FightingICE has held every year from 2013 up till now. It consists of 6 rounds, the composition of 2 leagues and 3 characters. The ranking of competitors will be determined by the total result of 6 rounds. The maximum fighting time of a round is 60s. After rounds 1 or 2, the characters' positions and HPs will be reset, and a new round is started.

The FightingICE environment features two players starting with a certain amount of hit points(HP) and the fights terminate when one of the players reaches zero HP or preset time runs

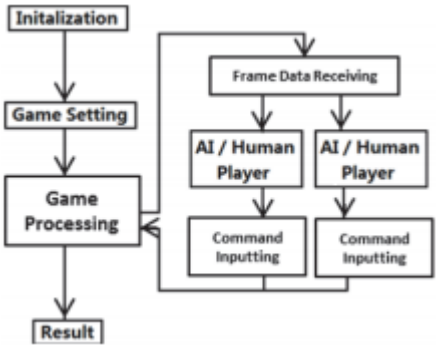

Fig. 1. FightICE game engine's architecture and the processing flow

out. Characters creation allows an opponent with predetermined move sets or randomized black-boxed move sets to support varying machine learning tasks. To better simulate human behaviour, frame information extracted by AI agents will also experience 15 frames or roughly 0.2 seconds of delay via platform. AI agents should act based on this inaccurate information. An extra speed run mode that encourages timely victory is implemented as well to support more complex AI fighting styles.

The players are represented by one of the three game characters ZEN, GARNET and LUD, each of which can perform certain skills. However, every character has different requirements for the skills and achieves different effects with them. This information is saved in the so-called character data. At the time of writing, only the character data for ZEN was available on the fighting game AI competitions website.

Besides the character data which do not change over time,the agents can access the so called frame data which change with every frame. This data contains information about e.g. positions of the two characters and the amounts of their health and energy points and are used by most of the existing agents for decision making. [8]

Here we have a snapshot of the gameplay provided by FightingICE in Figure 2.

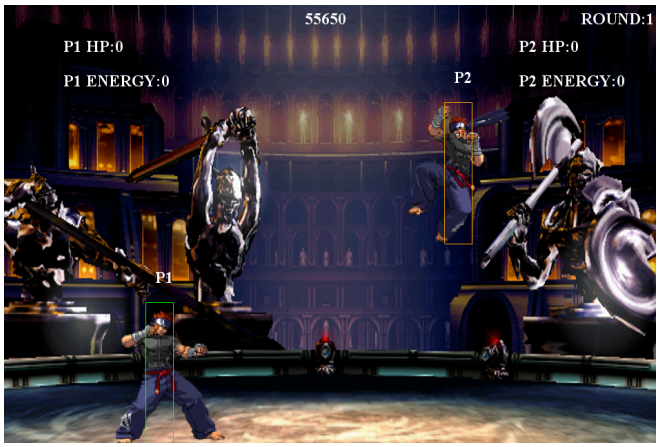

Fig. 2. A scene from FightingICE battle between two opponents

B. AI agents in Fighting Games

Fighting games have inspired multiple machine learning frameworks being implemented onto AI agents. Ruled-based bots with human devised play style constraints to match human level of playing showcase a promising start [9]; Yoon and Kim [10] used Convolution Neural Network(CNN) to train visual frame snapshot from the game to learn enemy pattern and execute offensive or defensive moves.Their input data is represented by 96×64 pixels image,and DQN is applied to the agent.

Monte Carlo Tree Search (MCTS) emerged as a trend in 2016 and has gained the spotlight of fighting games AI training. This preference is reflected from the dominating amount of work submitted to the Fighting Games AI Competition in recent years involving MCTS on top of previous trained agents since its annual held from 2013.

Reinforcement learning also received wide approval from community to train AI agents.Complicated and massive gaming environment hinders its performance on earlier platforms while its revitalized by the increase in computation power and emerging tools like MCTS and Deep Neural Network(ANN).

Dae-Wook Kim et al. [11] has evaluated and proven the combination of reinforcement learning and deep neural network are able to handle high complexity with approximation. It has surpassed human-level performance in fixed game environments and turn-based two player board games. They mix up MCTS and self-play in a 1:3 ratio and validates that it's able to overwhelm other AIs in the game with a high win rate. Joao Ribeiro Bezerra et al. [12] also corroborates the effectiveness with such combinations of the two.

In [13], Matheus R. F. Mendonc¸a et al utilized multiple types of reward functions alongside ANN. Varying play levels and styles from trained AIs using different reward functions suggests the significance of devising rewards.

Inseok Oh et al. [14] presents a practical reinforcement learning method that includes a novel self-play curriculum with different types of agents and data skipping techniques to facilitate explorations in vast spaces. Their work signals more human knowledge should be incorporated into reinforcement learning framework to create near pro-level player.

All agents described so far share in common that they assume agents only should take and learn atomic actions in any given state, while chains of actions into combos may prove to be optimal strategy. Gianlucca L. Zuin et al. [15] address the problem of neglecting combos in fighting games AI training. Combos are an integral part of many action games, rewarding precise execution of commands and encouraging players to keep on practicing. Competitive fighting game players strive to perform long combos that take the most out of any of their opponents' vulnerable moments. Such behaviour however is mostly absent in previous trained agents. Despite evaluation of above agents against AI opponents or even NPC opponents returns promising results, match against pro human player is rarely conducted and lacking data as well. More approaches still need to take into account an optimized human-level AI agent player.

III. METHODS

To address the problem of combos generating inhuman behaviour, we propose moderate changes in current reinforcement learning frameworks for fighting games AI training to realize combo mastering.

The variables in implementing our agents generally involve utilizing genetic algorithms to discover long strings of combo, defining states of the agent, determining possible action performed by agents in every state,specifying reward function and the implementation of artificial neural network.

A. *A Genetic Combo Finder*

To successfully learn combos, we first need to pre-process the game move set and identify possible and effective combos. As [15] suggests, an evolutionary algorithm allow us to trim the exponentially large search space of attack combinations. The main idea is to use a genetic algorithm to evolve a population composed of sequences of actions and, using an adequate fitness function, select the ones that are more suitable to be considered combos based on combo length, relative position in the action strings and potential chaining into future combos. Giovanna Mart´ınez-Arellano et al. [16] further showcases the availability of Evolutionary algorithms in a fighting games' setting. The returning results constitute as our combo pools which is integral to following steps in the training process.

We start by introducing the game platform that we used to find the combos (Subsection A.1), then the AI agents we used to gather the information for the genetic algorithm (Sub-section A.2). The action strings will be generated at the start of the game (Subsection A.3) and are evaluated by the fitness function, which identifies the combos at the end of the round(Subsection A.4). After fitness evaluation, actions are selected to undergo crossover and mutation (SubsectionA.5).

1) Game Platform: As we mentioned earlier, Fighting-ICE is the game environment we have chosen to experiment with our AI agents. The platform supports Java programming and Py4j can be used to develop AI agents in Python. The game platform sends the game state of each frame to the agents, which contains a large amount of information, including the positions of both players, the action an agent chooses, the motion an agent is currently executing, and the status of both players. The combo scheme is similar to Street Fighter, and it includes directional commands and two buttons for attacks and one for blocking. The attack may change depending on the player's current state, including the special moves that require the directional input before the attack. Each agent gathers energy from attacking and receiving attacks used to execute special moves such as throwing or shooting fireballs. The FightingICE has three different characters and supports 56 different actions for each character. The rules in detail are described in [17].

2) AI Agent Settings: In the game two AI agents fight each other by sending the action they want to use each frame. FightingICE accepts two types of inputs: 1) a series of inputs or 2) the name of the action. For example, we can let the agent send a series of commands like" 2 3 6 A", which tells the agent to perform a series of moves for the next few frames, or we can use the name of the action like "STAND D DF FA", which give us the same result as using the first method.

The genetic algorithm requires an agent to perform a series of actions and uses the score evaluated by the fitness function to decide the changes of the actions. We need two different AIs to gather the information we need. The first AI is called the PlayerAI, which takes a series of actions as input and performs the actions provided by the action list for each frame. Another AI is called the IdleAI, which does not perform any action at all. IdleAI records the frame it gets hit and outputs the frames it got stunned by PlayerAI to a file. IdleAI makes us have a smaller search space since it remains idle during the whole test. Because the actions might have a different results if IdleAI is in different statuses, for instance, if IdleAI is jumping when PlayerAI performs an uppercut, the attacks that hit him have a different result. This is also true if two players are in different positions when the combo starts. Another reason to keep IdleAI from moving is that being in the middle of an action counts as the character being uncontrollable. This would conflict with the data we are using from IdleAI.

3) Action String Generation: During PlayerAI's initialization, it will check for a file containing a map of action strings. There will not be one to begin with, so PlayerAI will create a new map object. The keys are based on variables that make up the state. The variables used as keys are the X distance, Y distance, and amount of energy PlayerAI has. X distance is separated into close, medium, and far distance. Y distance is separated into ground, mid jump, and apex of a jump. Energy is separated into none, very little, some , and a lot. This structure is showcased in equation 1. Each possible combination of these attributes will store a randomly created list of actions which we call the action string.

$$mapObject[xdistance][ydistance][energy] = actionString \tag{1}$$

4) Fitness Evaluation: To evaluate the fitness of each string of actions, both PlayerAI and IdleAI are used. PlayerAI executes the list of actions and IdleAI records the frames that it was stunned for during the round. The data IdleAI stores is in the form of '1's and '0's. Whenever there is a '1', IdleAI could not act on that frame. IdleAI stores this stun data in a file, which PlayerAI extracts the data from afterward. PlayerAI then splits the data into sections corresponding to each string of actions executed during the round. For each section, the longest string of '1's is found. The beginning and ending of the '1' string is recorded and the direct left and right string of '1's is measured. This formula is used to find the fitness of each side in equation 2:

$$sideFitness = ones/(1 + zeroes)^2 \tag{2}$$

The 'zeroes' in the formula are the zeroes between the longest string of '1's and the side strings of '1's. The total fitness value is calculated using equation 3:

$$fitness = leftFitness + rightFitness + longestString \quad (3)$$

The final version of the fitness function in equation 4 considers the total amount of time that IdleAI is stunned as well. This was done in hopes that more stunning would result in a bigger fitness value in equation 3 eventually. Equation 4 is weighed heavily in the favor of the result of equation 3 to ensure that longest stun time is considered above all else.

$$weighedFitness = (fitness*0.9)+(totalStunFrames*0.1) \quad (4)$$

5) Action String Mutation: Following each evaluation of an action string, there will be a check for a previous fitness value. If there was none, the current action string will be stored and a new child string will be created by mutating the current one. If there was a previous fitness value, the action string with the higher fitness value will be stored and used for the child. The child is generated by copying the action from the parent string with a 90% chance or randomly choosing an action with a 10% chance for each action in the string.

B. *Reinforcement Learning*

Reinforcement learning refers to an artificial intelligence technique where an agent learns to solve a given problem through its experience of past actions. Agents execute actions to move from state to state and a reward is given to the agent based on last actions taken. This action can be either positive if the action performed was beneficial to our predefined goal or negative otherwise.

1) State: A State reflects the situation of the agents in the gaming environment and generally encodes information necessary to the learning process in the reinforcement learning framework. In FightingICE features are HP points, energy remaining and relative distance between two players. With the addition of combos, a given state should also record the relative position in a combo string and potentially opponents past moves. This introduces temporal information in a state, thus enabling the AI agent to chain multiple actions together. Therefore we define the following features in a state:

- The HP values of the character player and its opponents. The value is normalized to [0,1] region. Agent's hit point will highly influence defense strategy and opponent's hit point can shape offensive methods applied.
- The character state of the agent. It consists of one of four types: GROUND, STAND, CROUCH and AIR. Different states provide agents of different sets of controllable actions that could be extended to the current state. A one hot encoding is implemented to represent these four discrete states.
- The relative distance between the two fighters. Attacks like punch, kick or defense actions like guard from both sides all have corresponding hit boxes. Only valid hit box collision lead to hits and influence HP value. Instead of recording all two-dimensional coordinates of both players or current hit box coordinates, distances between the fighters is able to convey possibility of collision and contribute to the effeteness of actions performed.
- The energy remaining of both sides. Energy is consumed to perform special skills in the game like throwing fireballs in contrast to basic actions. Since different skills require different energy consumption, this feature will affect how skills is incorporated into the attacks.
- The actions performed by both sides. It's not only a input feature of the state but the output of the learning process as well, which becomes next action taken.This is where the outcome of our former genetic combo finder come into existence. Instead of determining moves solely from the basic move set defined in game(illustrated in Figure 3), we also draw from the combo pool we created from the combo finder. The long combo with sequences of actions that we discover is also represented as one action available in a given state. A combination of the two with one hot encoding becomes our input space. However in order to enable long sequence of input during state shifting, we need the assistance of two following feature.
- Combo serial number. It denotes the serial number of the current action in selected long combo. With the help of this flag, we'll gain the knowledge of following action in the combo.
- Combo termination flag. The flag will be set to one if the current long combo reach its last input action or combo is forced to terminate due to staggering by opponents. A controllable flag function given by the game environment could help determine termination situations.

The above features of one frame constitute as a state information vector.

Base – Actions that are used in the normal status.			
STAND	CROUCH	AIR	
Move – Actions that can change the character's position.			
FORWARD_WALK	DASH	BACK_STEP	JUMP
FOR_JUMP	BACK_JUMP		
Guard – Actions that the character uses for protecting itself and reducing damage from the opponent.			
STAND_GUARD	CROUCH_GUARD	AIR_GUARD	
Recovery – Actions that arise when the character is hit or performs a landing.			
STAND_GUARD_RECOV	CROUCH_GUARD_RECOV	AIR_GUARD_RECOV	STAND_RECOV
CROUCH_RECOV	AIR_RECOV	CHANGE_DOWN	DOWN
RISE	LANDING	THROW_HIT	THROW_SUFFER
Skill – Actions that can generate attack objects such as a projectile (fire ball). These skills have 3 parts from start to the end: Startup, Active and Recovery.			
THROW_A	THROW_B	STAND_A	STAND_B
CROUCH_A	CROUCH_B	AIR_A	AIR_B
AIR_DA	AIR_DB	STAND_FA	STAND_FB
CROUCH_FA	CROUCH_FB	AIR_FA	AIR_FB
AIR_UA	AIR_UA	AIR_UB	STAND_D_DF_FA
STAND_D_DF_FB	STAND_F_D_DFA	STAND_F_D_DFB	STAND_D_DB_BA
STAND_D_DB_BB	AIR_D_DF_FA	AIR_D_DF_FB	AIR_F_D_DFA
AIR_F_D_DFB	AIR_D_DB_BA	AIR_D_DB_BB	STAND_D_DF_FC

Fig. 3. A list of basic actions provided by the gaming environment available to the character

2) Reward function: A reward function is capable of influencing the training process drastically [13]. A reward function determines the goal we want to reach and give positive

feedback to actions that are more likely to satisfy our goal. Firstly in a fighting game, the victory of the game should serve as our lowest requirement. Thus when the agent is defeated at the end of the round or reach timeout of the match, it'll receive a -10 reward while winning grant a +10 reward. Secondly, when the agent is able to damage the player, the agent should receive a positive reward. The reward value is normalized to [-10,10] region relative to our total health. Reward scale of 10 has been proven effective through trial and errors [11]. Thirdly,we also want to encourage player to utilize more long chains of combos like human pro player. After each round, we calculate the average number of hits land consecutively without taking damage, and reward the agent when it reach above a certain threshold value. Observing the demo fights provided by the competition, we set this threshold value to be 2 and one more combo attack increase the reward by 1 and vice versa.

3) Proximal Policy Optimization(PPO): Proximal policy optimization (PPO) algorithms have an advantage for efficient learning by inferring the value for each state and relative action value. Policy clipping also helps to adjust the policy stably without sudden shift. With continuous feature values like distance and energy, PPO is regarded to be more suitable than a Q-learning based algorithms [11]. Figure 4 illustrates the idea behind it.

Algorithm PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

Fig. 4.

With the experience of previous studies, the following table I showcases our environment setting for our reinforcement learning process.

4) Artificial Neural Network(ANN): Our ANN (illustrated in table II) is composed of multi-layer perceptron (MLP) with fully-connected layers.A state vector obtained from the game environment becomes an input of our neural network. The outputs are possible actions of size 66 that the agent can select. In our case, it's the combination of 56 basic actions in the game and 10 discovered combos from the genetic combo finder. Since we require the current combo to be continued, if the current step is part of the combo in the pool, the next actions in the combo action string will replace the output of the ANN unless the combo is terminated. We also adopt an action masking mechanism which filters the actions that are not possible in the current state. For example, an air projectile attack action is only valid when the player is in the air. When this action occurs from the output of network, it is transferred to the game simulator as a no-op action.

Name	Value
step size	4
batch size	64
learning rate	value 2e-4
value function coefficient	value 0.2
entropy coefficient	value 1e-3
clipping range for policy	0.05
clipping for value function	none
clipping for gradient norm	0.4
trade-off factor for generalized advantage estimator	0.95
minibatch size	value 4
number of epoch for optimizing surrogate	value 4

TABLE I
HYPER PARAMETERS FOR REINFORCEMENT LEARNING

Name	Value
minibatch size	32
learning rate	0.001
activation function	ReLu
neurons in hidden layer	[100,30]

TABLE II
HYPER PARAMETERS FOR ANN

IV. RESULTS

The performance of the genetic combo finder and reinforcement learning agent were evaluated individually.

A. Genetic Combo Finder

To evaluate the combos generated by the agent PlayerAI, 500 rounds were played against IdleAI. PlayerAI executed on average two action strings per round. At the end of each round, the fitness value of each action string was calculated. Every 10 rounds the data was added to a file. The data consisted of the 10 best action strings found during that round. The previous action strings are retained for the purposes of comparison. Figure 5 shows the change in the fitness value of the best action string every 10 rounds. From the data gathered, we can see that there are periods where the action string improves very quickly, and long periods where the action string does not improve at all. The combo finder was run for about 4 hours, and about half of that time was spent with no improvement.

B. Reinforcement Learning

We evaluated our agent by playing against other AIs. The opponent bots for evaluation were chosen from top rankers submitted in the 2019 and 2020 competitions. Three opponent fighters are named as SampleMctsAI, HaibuAI and TeraThunder. We fight 50 games with every enemy AI agent with a total of 150 rounds. The agents only played as character ZEN and started with 400 max health points (HPs). The game terminates until one of the player reaches 0 HP or the match timeouts at 60 seconds where the player with higher HP is the winner.

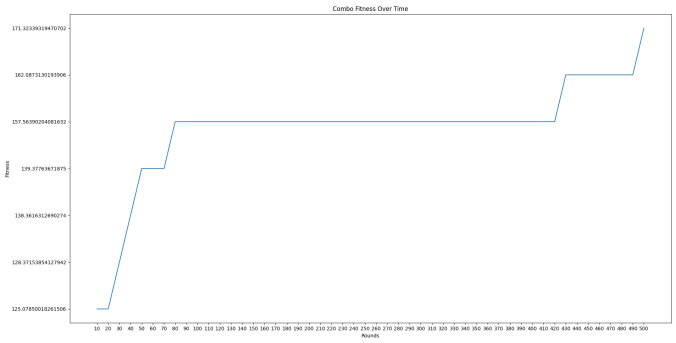

Fig. 5. Graph evaluation of the best action string generated by PlayerAI

To further test the effects of our agents compared to a traditional reinforcement learning agent, we'll set up a control group. We select a sample Deep Q-learning bot from https://github.com/ichara/DL and run the experiments with the above 3 opponent bots. The variables differ in:

- Agents only take actions from the basic action list (no long combo provided).
- Agents state features only contain the action, the HP values and relative distance of two players.
- Reward function is only related to win rates and HP value.
- Q-learning serves as the reinforcement learning algorithm instead of Proximal Policy Optimization.

The rest of the experiment configuration remain identical.

1) Win rates: Figure 6 and Figure 7 show the game results for both our agent and the simple DQN agent. As we can see, our agent achieve a higher overall win rates against three opponent bots compared to the sample agent. Our fighter is able to mostly win more than half of the match while the sample agents have a win rate of below 50%. However this discrepancy is rather minor. The emphasis on long combos for our agent only contribute very little to the general win rate.

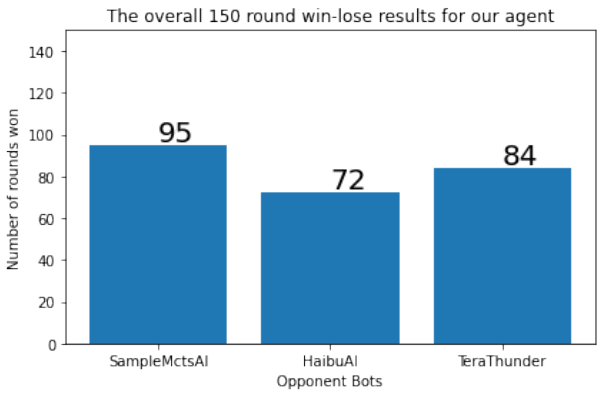

Fig. 6. Number of games won by our agent against opponent AIs in 50 games (150 rounds)

2) Combo length: Next, we also recorded the number of consecutive hits landing on opponents in every match. These statistics could reflect whether and how often the agent

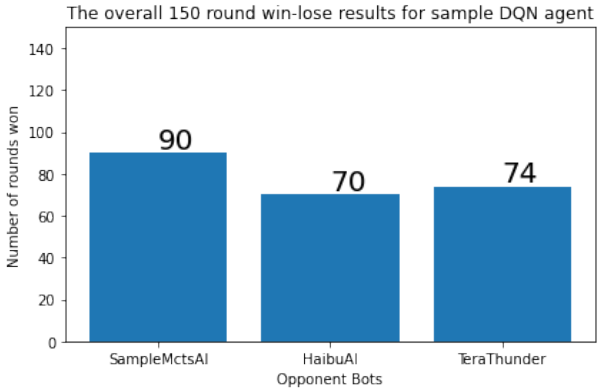

Fig. 7. Number of games won by sample DQN agent against opponent AIs in 50 games (150 rounds)

was able to fully or partially execute long chains of actions intended.

Figure 8 and Figure 9 shows the average numbers of chains of combo-attacks executed by AI agents throughout all of the experiment matches. We could notice that majority of the chained combo attacks length are of 1-3 and rare cases we could reach 4+ length, for both agents. It's a clear sign of the sheer difficulty for AI agents to execute long sequence of actions. This four division clearly indicate the level of persisting combos: 1 being just a singular and isolated action or combo being immediately interrupted; 2 or 3 indicates a a relatively short sequence of actions and 4+ represents a long combo. Comparison between two graphs clearly reveal that our implementation extend the average length of combos drastically. Atomic action of length 1 appears much less frequent with our agents and combos with varying length appears much more frequently.

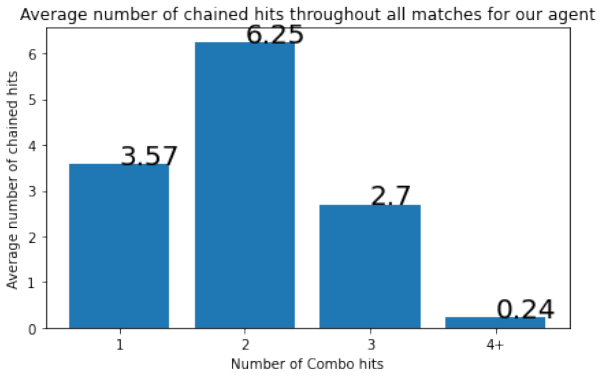

Fig. 8. The average number of successfully performed chains of combo-hits of the length 1 – 4+ for our agent

We notice that the previous win rate heuristics doesn't experience a major improvement with our agent implementation compared to combo length recorded. It indicates a non linear relationship between winning condition and ability to execute full or partial long combos, as we expected. Even though pro human player regards combo as high level of performance

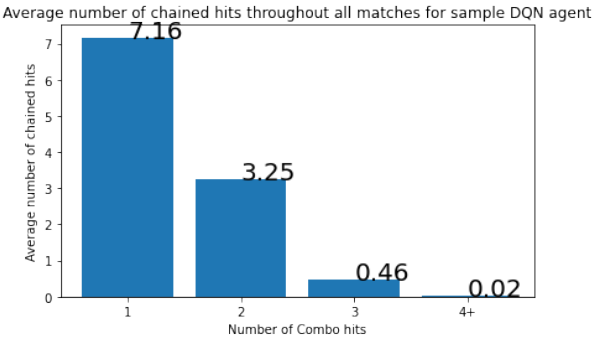

Fig. 9. The average number of successfully performed chains of combo-hits of the length 1 – 4+ for sample DQN agent

and in integral part of winning, the reverse can't be deemed absolute.

V. Conclusion and Future Work

In this paper, we identify the deficiency of combos in current fighting games AI implementation and propose modifications to existing customized reinforcement learning framework for fighting games and the addition of a genetic combo finder to master combos. The evolution generation test shows the feasibility for our genetic combo finder to discover long combos despite lack of refinement; The experiment with our agent successfully surpass the win rate and average combo length achieved compared to a non-special agent.

The main idea of our framework is a hierarchical implementation that separates the process of discovery of combo and learning through match against other players. The combo set constructed serve to be a fundamental part of the following learning process. Yet one may consider whether such hierarchy is a necessity for mastering combo and combos memorization and exploration could be merged into the learning process.

For the genetic combo finder section, one factor to improve upon is the level of improvement that the current algorithm achieves for combo generation. As stated above in the results for the genetic combo finder, there may be long periods of time where the action string does not improve. Some ways to better implement this could be to keep track of how much improvement the agent has achieved over time. With knowledge that the fitness is not improving, the combo finder could change the way new action strings are generated to potentially escape being trapped on one value in the future.

For the reinforcement learning section, we could potentially have many extensions on our current implementation. Different reinforcement learning hyper parameters could be retested and we can evaluate effects of hyper parameter on the training process; The state representation could also be more complicated and involve more features like the movement velocity or damage velocity of the player. A more efficient way of running the learning process like involving extra steps of MCTS could potentially mitigate the computation complexity accompanied with extra features.

The reward function in the learning process could be also related to a specific strategy, like "do not cast magic when far away from the enemy". [14] implies the significance of human knowledge. Hence, we could enforce a reward on predefined strategies that encourage action chaining, combined with our existing reward function. Human knowledge allows game developers to imbue some of their knowledge in the decision making process of the agent but still allow it to consider if these recommended actions are worth learning. If not, the agent can still adapt its ability and choose another strategy that suits it better in a given moment by using a secondary reward function. [13].

References

[1] Coffee-Stain-Studios, "Goat simulator," [CD-ROM], 2014.
[2] Capcom, "Street fighter," [CD-ROM], 1987.
[3] Midway-Games, "Mortal kombat," [CD-ROM], 1992.
[4] Bandai-Namco-Entertainment, "Tekken," [CD-ROM], 1994.
[5] Nintendo, "Super smash bros." [CD-ROM], 1999.
[6] ESPN-Esports-Staff. (2020) IEEEevo 2020: What happened to the biggest fighting game tournament of the year. [Online]. Available: https://www.espn.com/esports/story/id/29406401/evo-2020-happened-biggest-fighting-game-tournament-year
[7] F. Lu, K. Yamamoto, L. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas, "Fighting game artificial intelligence competition platform," 10 2013, pp. 320–323.
[8] X. Neufeld, S. Mostaghim, and D. Perez-Liebana, "Htn fighter: Planning in a highly-dynamic game," in *2017 9th Computer Science and Electronic Engineering (CEEC)*, 2017, pp. 189–194.
[9] (2013) Welcome to fighting game ai competition. [Online]. Available: http://www.ice.ci.ritsumei.ac.jp/ftgaic/index-R.htm
[10] S. Yoon and K.-J. Kim, "Deep q networks for visual fighting game ai," 08 2017, pp. 306–308.
[11] D.-W. Kim, S. Park, and S.-i. Yang, "Mastering fighting game using deep reinforcement learning with self-play," in *2020 IEEE Conference on Games (CoG)*, 2020, pp. 576–583.
[12] J. Bezerra, L. Góes, and A. Silva, "Development of an autonomous agent based on reinforcement learning for a digital fighting game," 11 2020, pp. 1–7.
[13] M. Mendonça, H. Bernardino, and R. Fonseca Neto, "Simulating human behavior in fighting games using reinforcement learning and artificial neural networks," 11 2015.
[14] I. Oh, S. Rho, S. Moon, S. Son, H. Lee, and J. Chung, "Creating pro-level ai for a real-time fighting game using deep reinforcement learning," *IEEE Transactions on Games*, vol. PP, pp. 1–1, 01 2021.
[15] G. Zuin, Y. Macedo, L. Chaimowicz, and G. Pappa, "Discovering combos in fighting games with evolutionary algorithms," 07 2016, pp. 277–284.
[16] G. Martinez-Arellano, R. Cant, and D. Woods, "Creating ai characters for fighting games using genetic programming," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. PP, pp. 1–1, 12 2016.
[17] (2021) Fighting game ai competition homepage. [Online]. Available: https://www.ice.ci.ritsumei.ac.jp/ ftgaic/index-1.html