TEXAS STATE UNIVERSITY

MEMBER THE TEXAS STATE UNIVERSITY SYSTEM

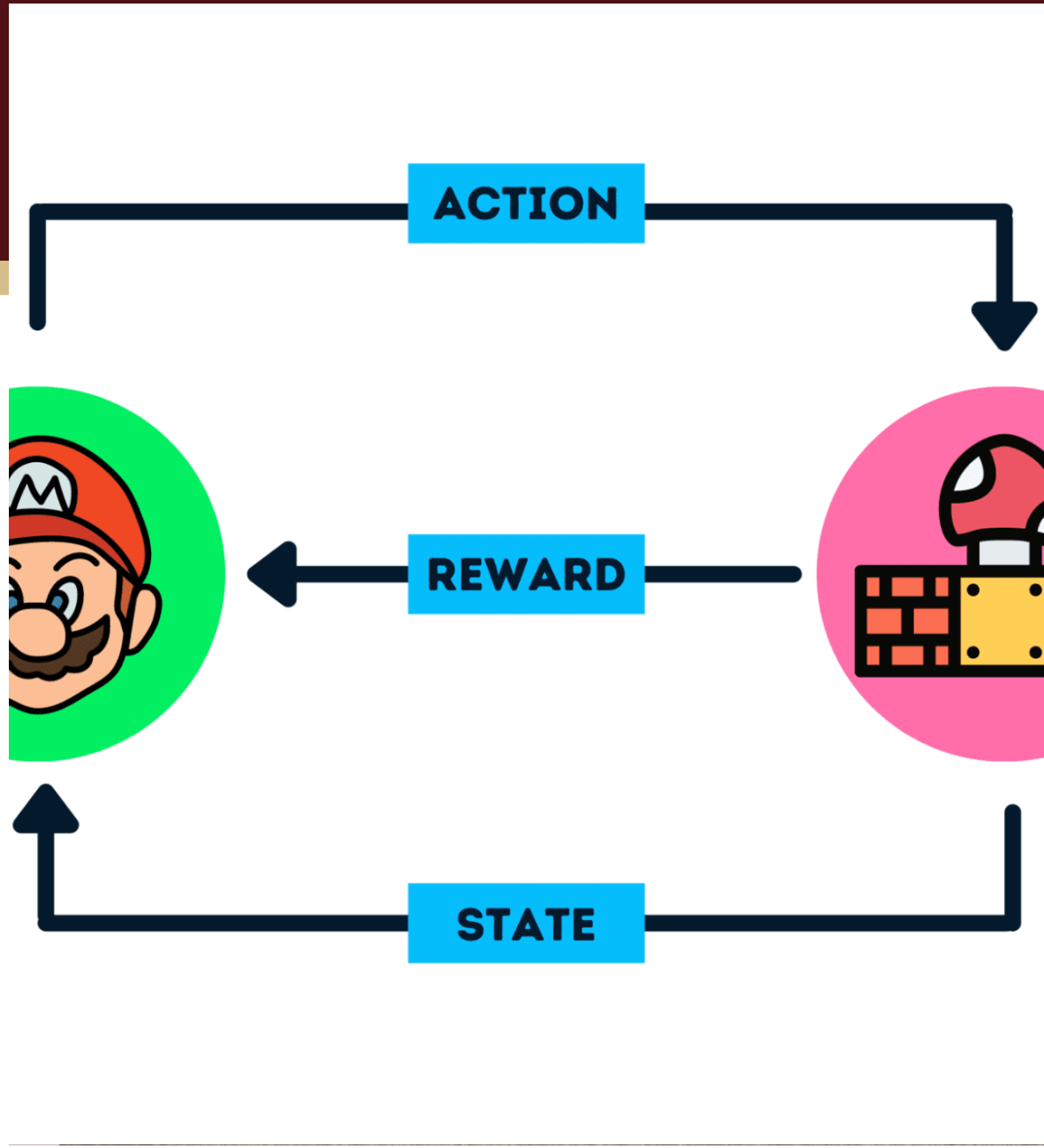
Paper Talk: “Go with the Flow - Reinforcement Learning in Turn-Based Battle Video Games”

Authors: Elinga Pagalyte, Maurizio Mancini, Laura Climent

Venue: IVA 2020

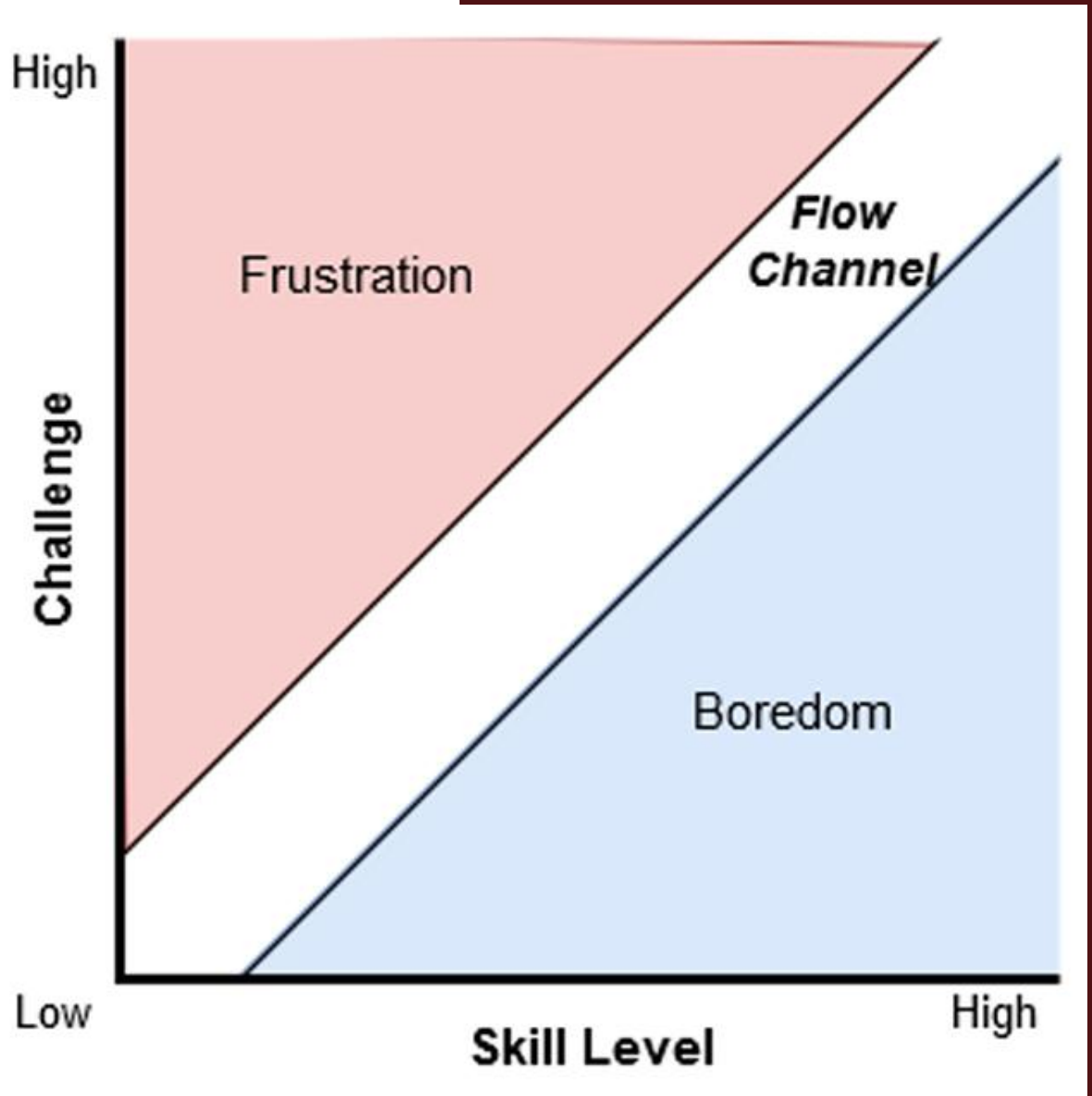
Presented by Joseph Nikhil Reddy Maramreddy

Presented as part of the Final Project for course CS 5325.001



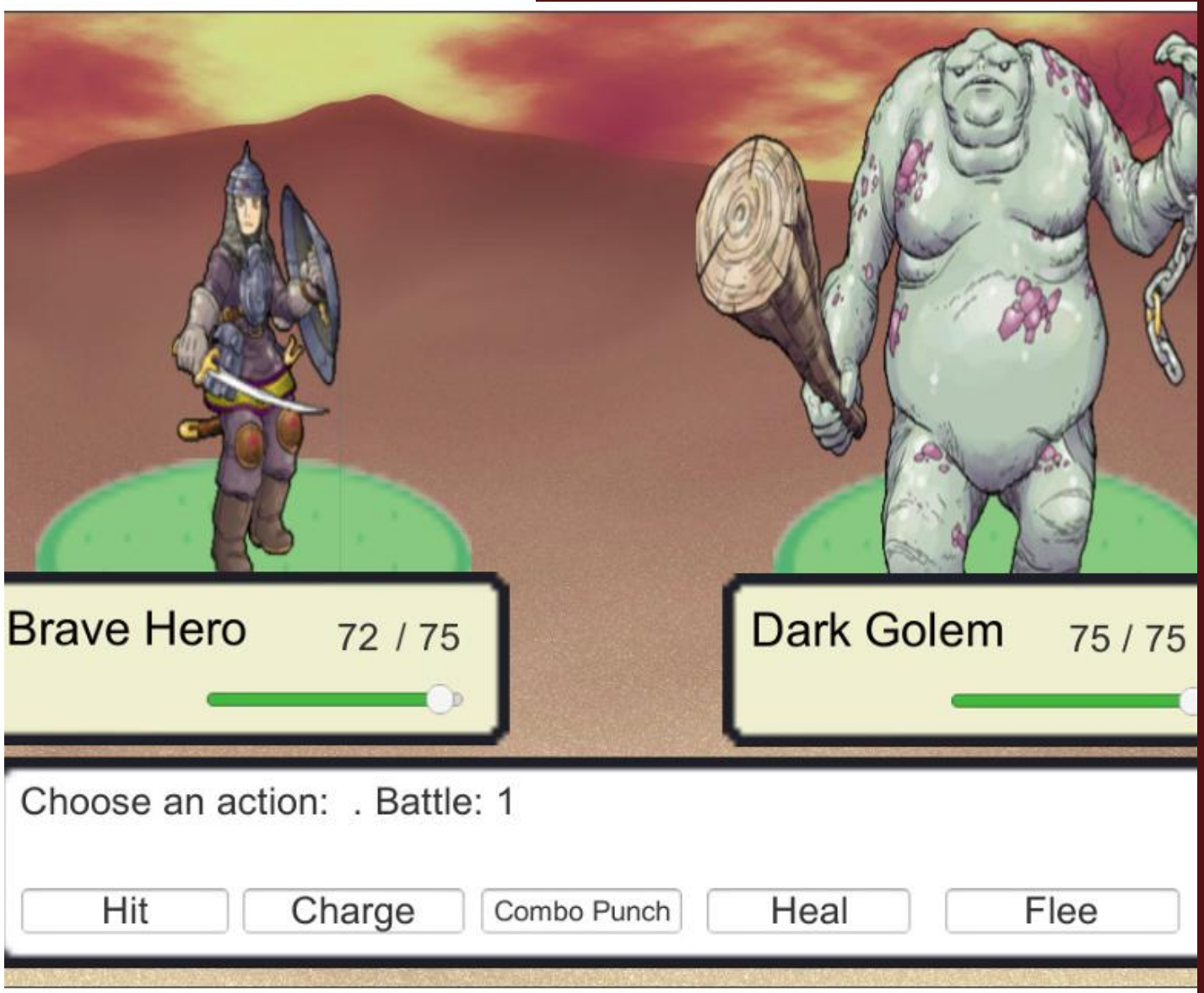
Motivation & Problem Statement

- Traditional games use fixed difficulty levels (Easy/Medium/Hard).
- These fail to adapt to each player's skill level.
- Too easy → boredom; too hard → frustration.
- Dynamic Difficulty Adjustment (DDA) aims to keep players in flow - a balanced challenge.



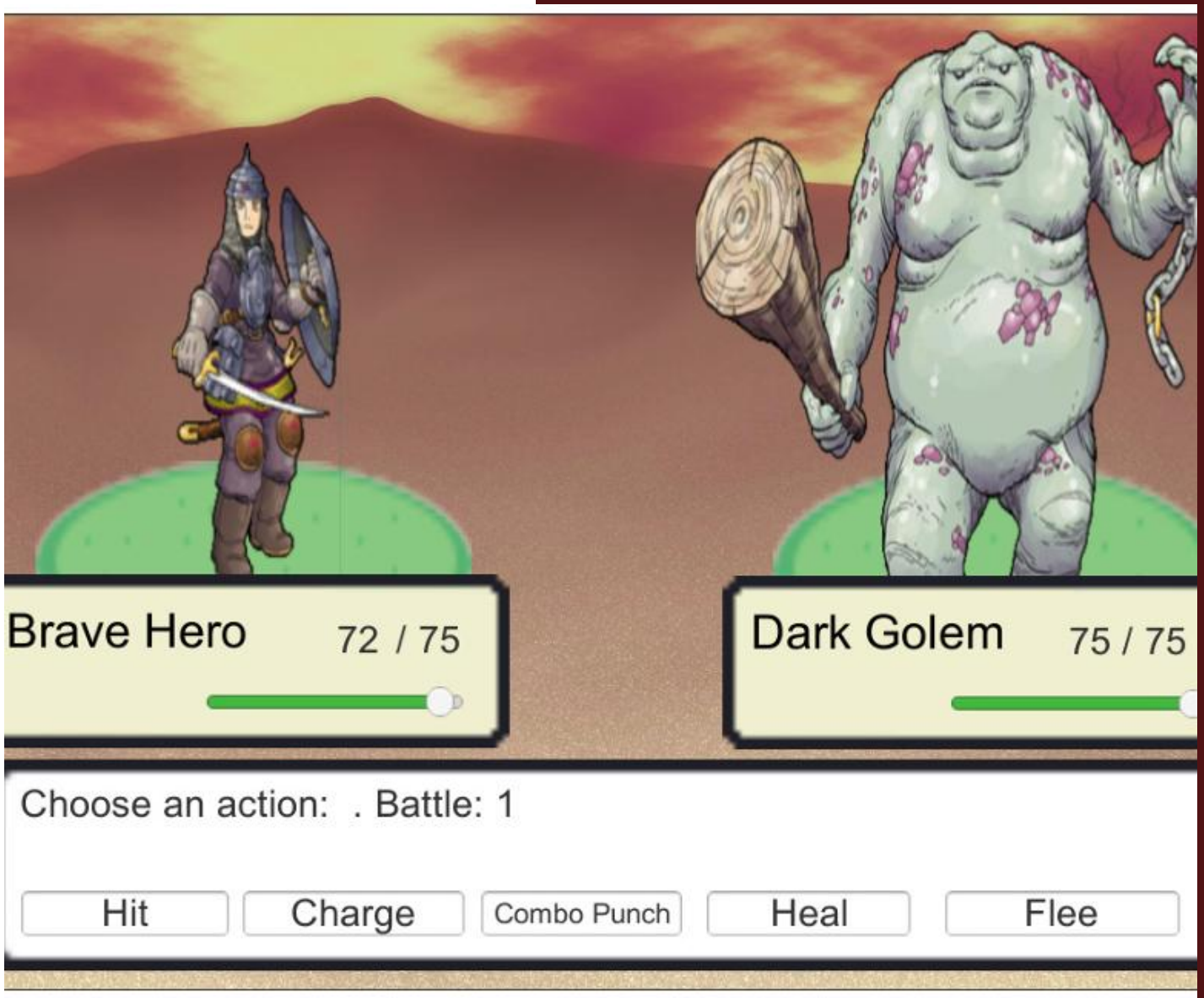
Approach Overview

- Built a 2D turn-based battle game (similar to classic RPG fights).
- Implemented a Reinforcement Learning (SARSA) agent as the opponent.
- Used RL to adjust gameplay dynamically (learning from ongoing battles).
- Evaluated both agent behavior and player engagement through experiments.



Game Mechanics

- Each side (player & agent) starts with 75 HP.
- Player chooses from 5 moves: Hit, Charge, Combo, Heal and Flee.
- The AI also has these moves (no Flee).
- Battles end when one side's HP ≤ 0.



Reinforcement Learning Setup

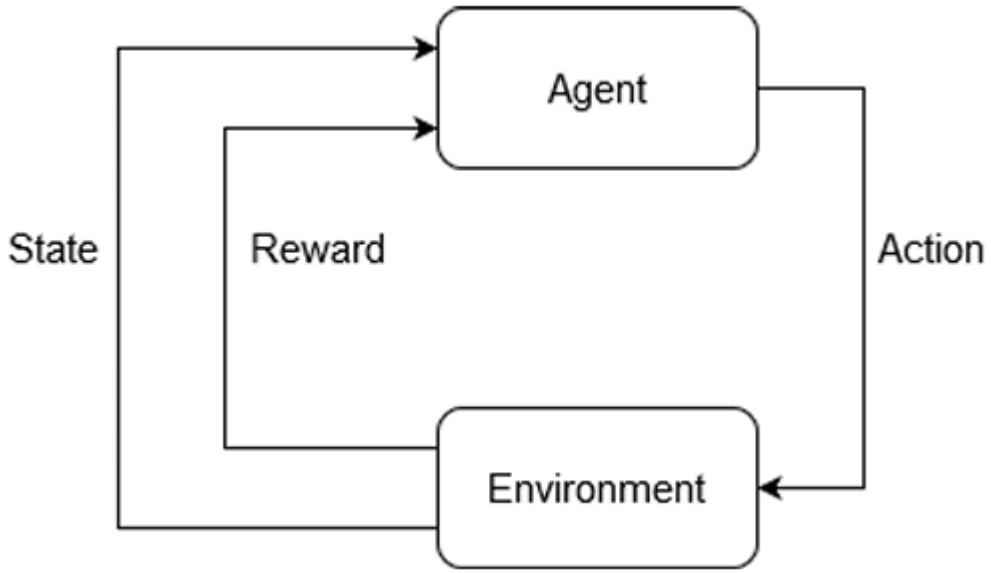
- Algorithm: SARSA (on-policy RL).

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$$

- States: 17 discrete HP-difference bands (e.g., equal HP, player ahead, AI ahead).
- Actions: Hit, Charge, Combo, Heal.
- Reward:
  - +100 for winning, −100 for losing (or ±50 variant).
  - Rewards swapped after 5 wins/losses → Dynamic Difficulty Adjustment.

Reinforcement Learning Setup

Diagram of the SARSA algorithm.



The states for the SARSA RL agent.

State	Player HP	Agent HP
0	75	==
1	[51-74]	==
2	[31-50]	==
3	[16-30]	==
4	[1-15]	==
5	<Agent HP	[66-74]
6	<Agent HP	[46-65]
7	<Agent HP	[31-45]
8	<Agent HP	[16-30]
9	<Agent HP	[1-15]
10	[66-74]	<Player HP
11	[46-65]	<Player HP
12	[31-45]	<Player HP
13	[16-30]	<Player HP
14	[1-15]	<Player HP
15	0	-
16	-	0

Experiment Design

- **Participants:** 4 players (2M, 2F, ages 20 - 40).
- **Games:** 10 game configurations (varying reward magnitude & exploration rate).
- **Each game:** 30 battles.
- **Measures:**
  - Player move usage
  - Agent's action evolution
  - HP trends per battle
  - Player feedback via **Game Experience Questionnaire (GEQ)**

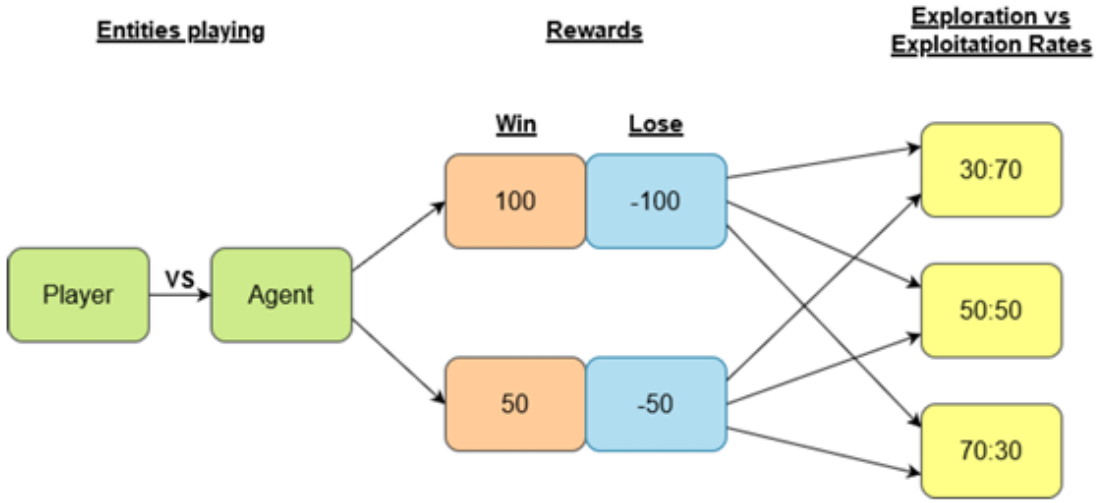


Figure 4: tree diagram with all possible parameters choices.

Results: Agent Learning Behavior

- Early training → agent overused Heal, leading to long matches.
- Later → agent learned to balance between attacking and healing.
- (30% exploration / 70% exploitation) gave best results - balanced challenge.
- Agent move distributions (combo, heal, etc.)
- HP traces over battles (showing more balanced fights).

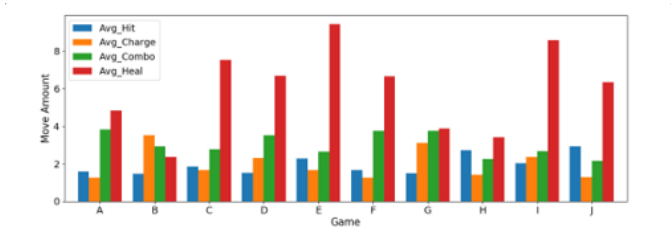


Figure 6: RL agent's average moves over 10 games (A-J).

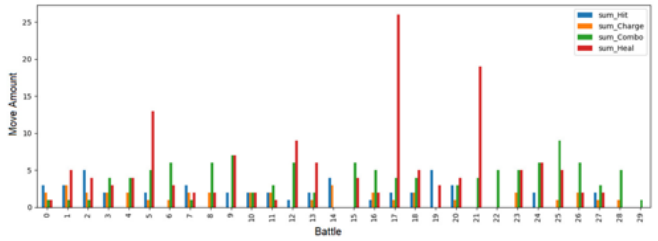


Figure 7: RL agent's moves sum in the 30 battles of game A.

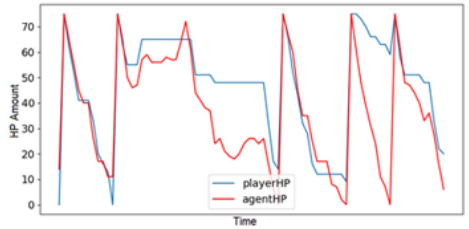


Figure 8: Player's and agent's HPs from battles $[16-20]$ of game A.

Results: Player Experience

- Measured via **GEQ** (challenge, competence, effort, flow).
- RL-based AI produced **balanced challenge and engagement**.
- Fixed Easy AI → boring; Hard AI → frustrating.
- RL opponents often scored highest for flow and enjoyment.

Gameplay type pairs	success	skillful	challenge	effort
easy vs hard	**0.03**	1	**0.001**	**0.001**
easy vs 50-50	1	1	**0.001**	**0.005**
easy vs 70-30	0.107	1	**0.001**	**0.001**
easy vs 30-70	1	1	**0.001**	**0.001**
hard vs 50-50	0.085	1	0.067	0.848
hard vs 70-30	1	1	**0.011**	0.107
hard vs 30-70	**0.002**	0.522	0.239	0.957
50-50 vs 70-30	1	1	1	1
50-50 vs 30-70	0.239	0.811	0.368	1
70-30 vs 30-70	0.1	0.811	1	1

Table 3: results of paired t-tests computed between all the combinations of gameplay types. Significant p-values ($p < .05$) after Bonferroni correction are highlighted in bold.

Limitations

1. **Small state space & simple environment.** Results may not scale to complex games.
2. **Reward swapping heuristic** (after 5 wins/losses) may cause abrupt difficulty shifts.
3. **Small participant pool** → limited statistical significance.
4. **Heal-spamming artifact** shows incomplete reward shaping.
5. **No comparison to other RL methods** (e.g., Q-learning).

My Implementation Plan

- Implement a simplified SARSA agent in Python with:
  - 17-state representation (HP difference).
  - 4 actions (Hit, Charge, Combo, Heal).
  - Reward swapping for adaptive difficulty.
- Simulate training and visualize:
  - Agent learning curve.
  - HP difference per battle.
  - Move distribution evolution.
- If time permits, I also want to make a small project using a game engine.

Feasibility

- Because the SARSA algorithm and state-reward structure are straightforward to implement in Python or Unity, the project requires only basic computational resources.
- The RL agent operates on a discrete state-action space with small-scale training data; hence, this project is computationally feasible without GPUs or large datasets.
- Any potential challenges that can arise during experiments, such as reward tuning or overuse of specific actions (e.g., Heal), can be managed through controlled simulation and logging.

Thank You!

Joseph Nikhil Reddy Maramreddy – yue15

CS 5325.001 – Reinforcement Learning

Citation: Elinga Pagalyte, Maurizio Mancini, and Laura Climent. 2020. Go with the Flow: Reinforcement Learning in Turn-based Battle Video Games. In Proceedings of the 20th ACM International Conference on Intelligent Virtual Agents (IVA '20). Association for Computing Machinery, New York, NY, USA, Article 44, 1–8. https://doi.org/10.1145/3383652.3423868