Final Project Status Update

AA 228 - Decision Making under Uncertainty

Stanley Wang (swang11@stanford.edu)
Steven Salah-Eddine (stevense@stanford.edu)

NOTE: *Our initial proposal hypothesized the application of decision-making techniques toward food recommendation. However, given the non-sequential nature of this problem, we have revised our project topic as follows.*

Introduction

Our project is inspired by the genre of games known as **battle royale** (Fortnite, PUBG, Warzone, etc.). These games pose a unique application for decision-making, where multiple agents compete against each other in a finite-horizon adversarial dynamic environment. The development of algorithms for this application can generalize far beyond video games–and can help us better understand generalized policies for adversarial multi-agent environments.

Environment Setup

We begin by creating a custom simulation environment in MATLAB (Appendix B) for testing our policies. Specifically, we consider a 50x50 discretized 2D grid world upon which our agents act. Thus, we have two states for spatial coordinates, $x, y \in [1, 50]$. In battle royale games, players compete with each other in a finite-duration, shrinking zone (typically imposed by a shrinking "storm" that damages players if they are caught in its area of effect). Thus, we consider a finite-horizon problem with t = 200 steps. However, to simplify our initial approach, we wrap the time step variable into an additional state $t \in [1, 200]$ and apply infinite-horizon learning methods. An implementation of this with a map inspired by Fortnite is given in Fig. 1

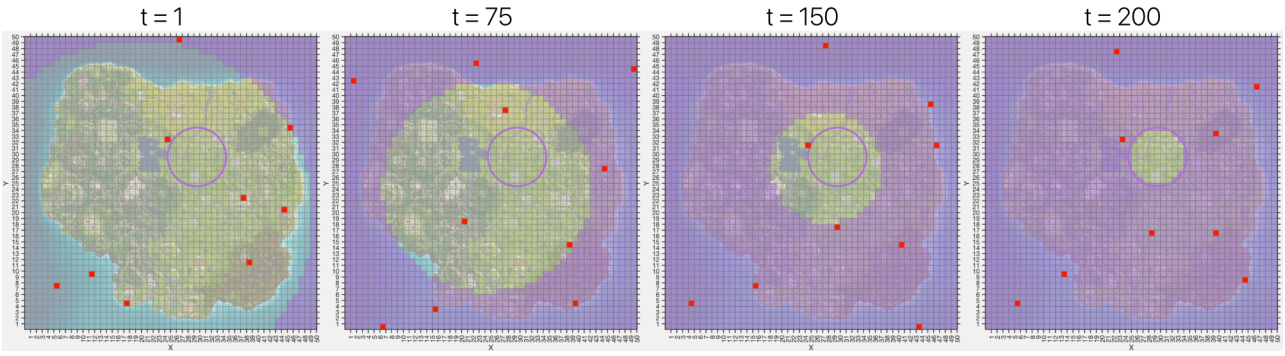


Figure 1: Simulation of Fortnite "storm" closure over a 200-step finite-horizon discretized environment. Agents (as marked in red) follow randomized policies.

Reward Setup

To summarize, we approximate our finite-horizon problem into an infinite-horizon problem with state variables $\{x, y, t\}$. Since x, y have 50 possible values each, and t has 200 values, there are $50 \times 50 \times 200 = 500,000$ states. The next step is to define the reward associated with any particular state, or $R(x, y, t)$. We construct the reward as follows:

$$R(x, y, t) = R_{map}(x, y) + R_{storm}(x, y, t)$$

$R_{map}(x, y)$ is the reward associated with the fixed topology of our "map". Logically, being out of bounds on the map (not on the battle royale island) carries a large negative reward. Certain locations on the map are "hotspots" (have better gear and loot) and thus carry a positive reward. $R_{storm}(x, y, t)$ is the dynamic reward associated with the characteristics of the storm at time step t. A negative reward is imposed if the player is in the storm, and the reward is zero otherwise. A linear gradient is imposed at the edge of the storm such that there are no discontinuities in R_{storm}. Examples of states with different rewards $R(x, y, t)$ are given in Fig. 2

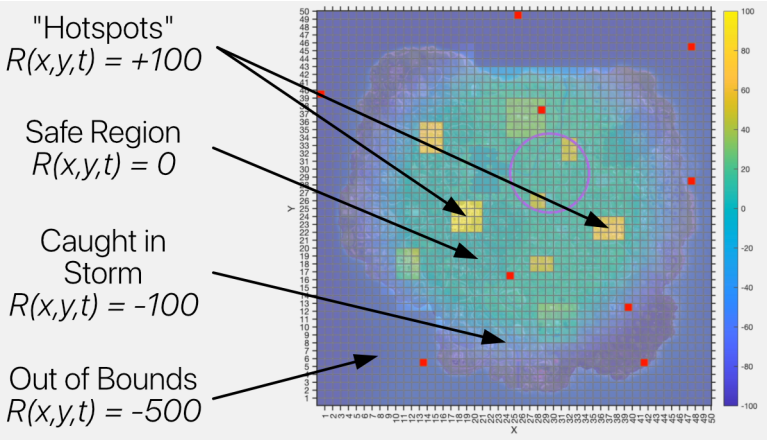


Figure 2: The reward $R(x, y, t)$ at a given state is dependent on map topology and the dynamic storm

Simple Q-Learning

Before considering the adversarial multi-agent case, we construct a single agent trained with Q-learning for navigating our environment. Intuitively, to maximize reward, the agent should spend as much time as possible at "hotspots" while minimizing time in the encroaching storm. We consider five actions $a = \{1, 2, 3, 4, 5\}$ corresponding to moving up, down, left, right, or staying in place. In our initial implementation, we do not introduce stochasticity in the transitions. For example, moving up ($a = 1$) will always result in $y^{(t+1)} = y^{(t)} + 1$. Finally, we assume a discount factor of $\gamma = 0.95$

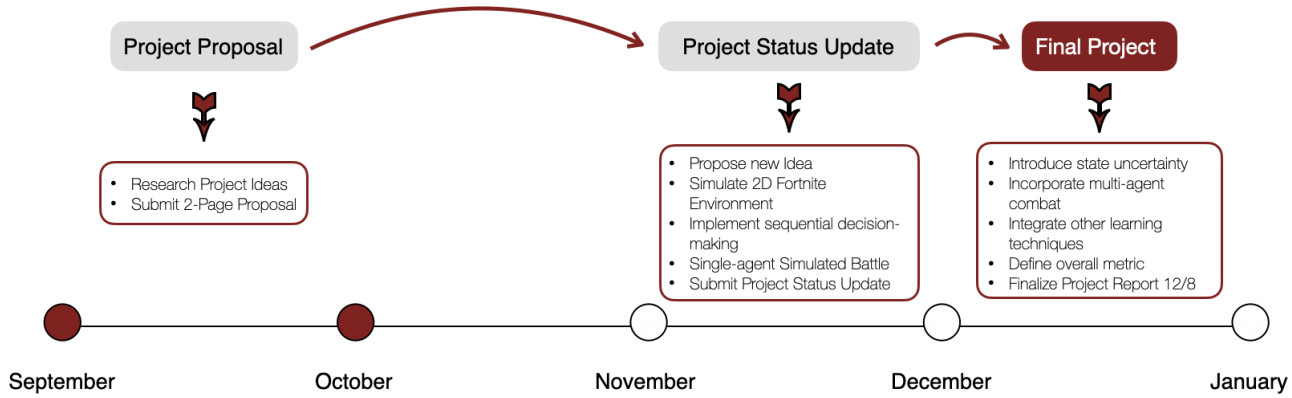
Transition data similar to Project 2 in the form of $\{s, a, R, s_p\}$ is generated by sampling random values for $s = \{x, y, t\}$ and a, then calculating $R(x, y, t)$ and the next state s_p after action a. In total, one million sampled transitions are generated at random. Then, Q-learning is implemented with a learning rate of $\alpha = 0.1$ and 250 passes over the sampled transition dataset. The resulting performance is surprisingly decent, with demonstrations provided in the Appendix A.

Next Steps

Our initial approach with Q-learning on a single agent in our simulated battle royale environment has been successful, as shown by Appendix A. This serves as an excellent baseline for subsequent development steps, which include but are not limited to:

- Introduction of **stochasticity** in state transitions. Introduction of state uncertainty?
- **Multi-agent interaction** (combat). The reward function can be manipulated to either encourage or discourage interaction with other agents.
- Exploration of **other learning techniques** besides Q-learning. Online methods? Pre-training with extensive artificially generated transition data from simulation may not be realistic.
- Development of an **overall metric** for evaluating the performance of different policies in our battle royale environment. Survival time? Total reward?

Project Timeline



Appendix A

Cool demos of our single-agent Q-learning policy:

https://youtu.be/1YOWFgyXa7Y?feature=shared

Appendix B

GitHub repository of MATLAB code for simulation and Q-learning training:

https://github.com/StanleyWang1/DefaultDancr