RAF-Deep RPG

Vanja Kovinjic, David Ilic, Jovana Radakovic

January 7, 2024

Abstract

This report explores the development of a reinforcement learning agent for the 2D RPG(Role-playing game) RAF-Deep-RPG. The objective is to design an agent capable of making intelligent decisions to optimize resource collection, avoid threats, and efficiently progress through the game levels. This report provides insights into the problem analysis, agent design, and the results of the developed reinforcement learning agent, along with a conclusion.

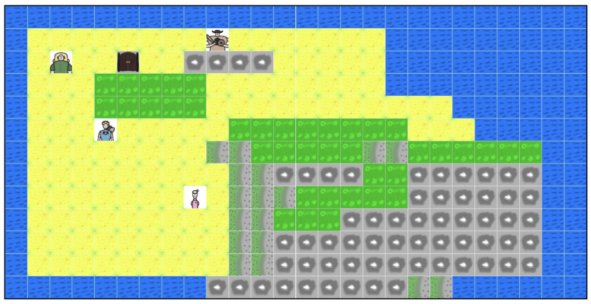


Figure 1: Example of a map

1 Introduction

RAF-Deep-RPG is a 2D RPG game, where the player's task is to reach the gate, pay the toll, and move on to the next level. The ways in which the player collects resources are various: collecting materials in the fields on the map and receiving rewards from the peasant. After collecting the resources, the player can sell them for gold, which is necessary to advance to the next level. The merchant, located on the map, serves this purpose. As a counterpart to the peasant who rewards the player, there is also a bandit who attacks the player and steals their resources.

This game requires the player to employ strategic thinking and decision-making skills. With diverse materials, characters, and varying levels of accessibility in different fields, the game offers a complex environment for an intelligent agent to navigate.

The primary goal of this project is to design a reinforcement learning agent capable of learning optimal strategies to maximize resource collection, mitigate losses from the bandit, and efficiently progress through the levels by paying the gate toll. This report outlines the problem analysis phase, where observations and conclusions are drawn from the game documentation, laying the foundation for the agent design.

2 Problem analysis

Analysing the game we have came to some conclusions and tactics that we have used to design our agent. We stated them as the following observations and conclusions. **Observation and conclusion #1:** Non-playable characters such as farmers, bandits and vendors add dynamic elements to the game. While not every map features a farmer, if present the agent should incorporate the presence and proximity of the farmer into its decision-making process as it is beneficial for the player to approach the farmer,since they provide random rewards. **Observation and conclusion #2:** The vendor has a maximum budget of 50 gold, and this is the most they can offer at any given time. Given you have resources which are all worth 7 gold each, the vendor can provide you with a maximum of 49 gold [as they cannot exceed their budget]. To continue selling, you must wait for a certain amount of time until the vendor replenishes their budget. Therefore, instead of aiming for exactly 50 gold, we aim to collect slightly more. This approach increases the likelihood that, when we approach the vendor, they can immediately give us the maximum available amount of 50 gold. Gathering a little more than 50 gold is strategic to enhance the chances of receiving the maximum amount from the vendor during each interaction. Additionally, by accumulating a surplus, we aim to improve the probability of retaining valuable materials with lower values. This precaution is nice to have because the bandit has the capability to randomly steal items and random quantities. If the bandit attacks when you have more than 50 gold, there's a chance you will still possess over 50 gold after the encounter, providing a buffer against potential losses. **Observation and conclusion #3:** Upon successfully selling the collected materials to the merchant and acquiring the desired 50 gold, engaging in further interactions with the bandit, farmer, or merchant becomes redundant. In order to optimize time the agent should be programmed to skip interactions with the bandit, farmer, or merchant after achieving the goal of obtaining 50 gold and should head straight to the gate. **Observation #4:** Every level has 13x26 fields which consist of the following:

- Level 0: characters:3 water:52 forest:22 hill:9 mountain=86 orchard:165 avg points: 2322

- Level 1: characters:4 water:103 forest:39 hill:17 mountain:62 orchard:112 avg points: 1402
- Level 2: characters:6 water:78 forest:31 hill:10 mountain:72 orchard: 140 avg points: 2024
- Level 3: characters:4 water:209 forest:9 hill:0 mountain:38 orchard: 77 avg points: 1032
- Level 4: characters:3 water:93 forest:15 hill:0 mountain:118 orchard: 108 avg points: 2104
- Level 5: characters:4 water:133 forest:53 hill:23 mountain:32 orchard: 92 avg points: 1406

Conclusion: Levels which have more points should be easier to obtain valuable materials and complete. We should implement an exploration-exploitation strategy to balance the agent's actions between exploring new fields and exploiting known fields to collect valuable materials. **Observation and conclusion #5:** Since there are 5 different levels in the game,we should train the agent over multiple epochs with varying map configurations to ensure adaptability to different scenarios.

3 Designing the agent

Our initial approach was to maintain a table that captures every conceivable state the agent might be in, along with the associated reward. This table essentially serves as the model. For instance, being in the top left corner without encountering a bandit results in 27 x 13 states; shifting one step to the right introduces another set of 27 x 13 states, and so forth. Realizing the impractical size of this table, we chose not to proceed with it. Instead, we opted for the neural network as a versatile function approximator, effectively mimicking the table's functionality. Inputting two coordinates each for the nearest unexplored field, explored field, bandit, peasant, and invalid field. This results in 10 inputs, capturing the agent's proximity to each element using Manhattan distance. The agent receives outputs in terms of rewards for various actions (e.g., moving up, left, right, down). With 5 possible outputs, the neural network internally computes this table based on the input and delivers the corresponding output. However, it might take some time for the model to discern which coordinates correspond to entities like the peasant or the bandit so we decided not to continue pursing this idea. Then we tried working with convolutional neural networks and switching our approach. Instead of training a single end-to-end model to learn the entire game dynamics, we've divided the task into three distinct parts. This approach aims to simplify the learning process by focusing on specific aspects of the game: gathering rewards, going to the vendor and lastly going to the gate. To do so we have 2 models:one specializes in learning to gather rewards, while the other focuses on navigating to specific locations

which is utilized twice: once to reach the merchant and later to approach the gate.

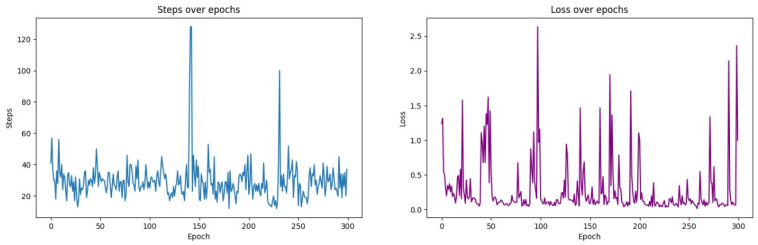
Both agents are equipped with CNNs each tailored to accept a 3x3 matrix as input. To decide what input the network gets, we use epsilon-greedy strategy. Epsilon starts high and decreases as the game progresses. Initially, we want to explore more and make random moves to understand the surroundings. As the game advances, we explore less because we're more familiar with the map. For the agent specializing in collecting rewards, the matrix encapsulates the surrounding fields. Conversely, the second agent's input matrix consists of "0", with "1" indicating the direction it should move in. To guide the agent, we check the map to locate the player, merchant,bandit etc. and run a BFS (Breadth-First Search) to find the best path, avoiding bandits. This path determines the reward for the model. Our strategy was when we receive a map, we consult the neural network to determine the next move. This process continues seamlessly as long as the game is in progress. During each iteration, we consistently receive updated maps and leverage a Breadth-First Search (BFS) algorithm to construct a 3x3 matrix. This matrix serves as valuable input for the neural network, assisting it in making informed decisions about the next move. This entire cycle unfolds within a loop embedded in the training files. In the training process, we balance exploration and exploitation. For example when training the second model we use epsilon of 0.4 meaning 40% exploration and 60% exploitation. We explore less as the agent learns more, playing more of the best moves it knows. This is the balance between trying new things and sticking with what works best. Additionally, the training involves the use of gamma coefficient, set at a specific value(in our case 0.2), which influences how much the agent considers future rewards.The higher the value the more it looks into the future. In the end we combined our models to form a multi-agent.

Parameter	Value
epoch	300
batch size	1
epsilon	0.3
epsilon decay	0.95
gamma	0.2

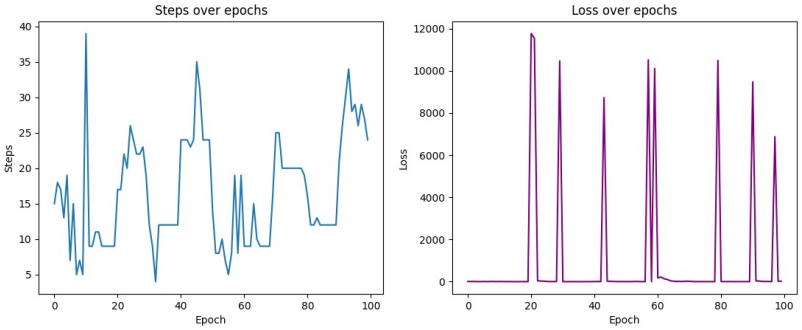
Parameters for model 1

Parameter	Value
epoch	100
batch size	1
epsilon	0.4
epsilon decay	0.8
gamma	0.2

Parameters for model 2



Model 1 training over 300 epochs with switching maps every 10 epochs



Model 2 training over 100 epochs with switching maps every 10 epochs

4 Conclusion

In conclusion, the training of an end-to-end model presents challenges, primarily due to the nature of modeling the rewards properly. Even when we break down the task into two distinct parts, explaining what we want the model to do is tricky. The complexity is further heightened when dealing with an environment that demands simultaneous actions, such as collecting rewards, reaching the farmer, and approaching the gate. Perhaps, in more complex environments, training agents to execute discrete actions independently before combining them may offer a more successful approach.