Catch The Quince

Hannah Bergenroth[1], Evelyn Bankell[2]

Abstract
This paper presents a simple 2D game that has been implemented in python, where the game idea is for a basket to catch falling quinces. An AI-agent is trained using a *Reinforcement Learning* algorithm called *Q-learning*. Q-learning is a model-free algorithm that explore to find the best possible action to take at each state. In this project, the actions are set to either move the basket to the left, to the right or do nothing and thus stand still. The goal of the agent is to learn a behaviour that maximizes the total reward over an episode. The result shows how the total reward increases with the number of iterations as well with applied Q-learning.

Source code: https://github.com/hannahbergenroth/CatchTheQuince

Authors
[1] *Media Technology Student at Linköping University, hanbe563@student.liu.se*
[2] *Media Technology Student at Linköping University, eveba996@student.liu.se*

Keywords: Artificiell Intelligens — Reinforcement learning — Q-learning

Contents

1. Introduction

The concept of Artificial Intelligence (AI) is making computers able to perform thinking tasks that humans are capable of [1]. The game presented in this report is agent-based AI, where an agent in form of a basket is placed at the bottom part of the screen. The basket uses information from the game data, that is based on the distance to the falling fruits (quinces). Whether the basket should take an action and move to the left, right or to do nothing is based on that information. The fruits are randomly generated at the top of the screen and can fall at two different speeds. If the basket catches a quince, the score increases and a new quince is generated, while the game starts over in case it misses. The agent is trained through a *Reinforcement Learning* algorithm called *Q-learning*.

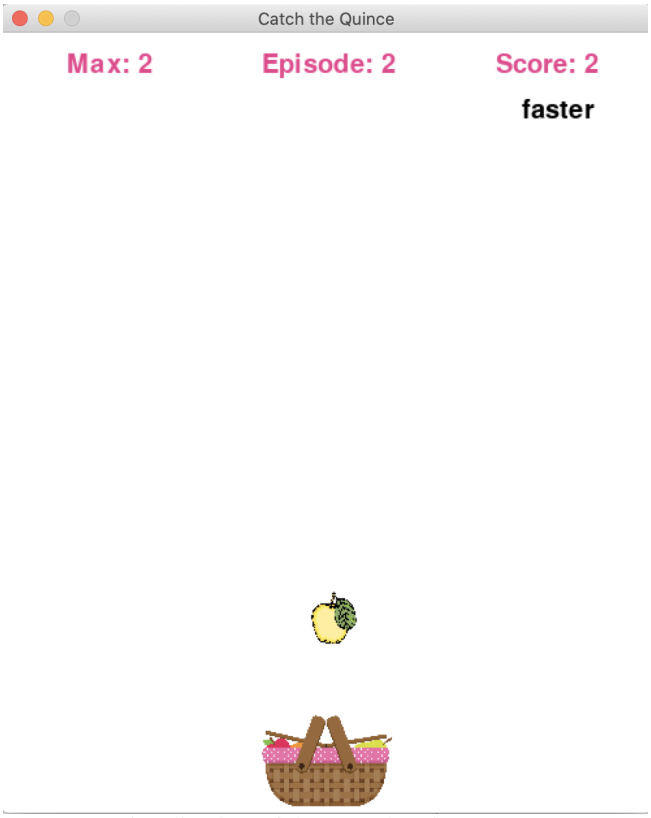

Figure 1. Visualization of the completed game implementation.

2. Theory

2.1 Reinforcement Learning

Reinforcement learning is an area within Machine learning, where an agent makes a decision based on its environment with the goal to achieve a maximized reward despite being uncertain about its environment [2]. Future states of the environment are affected by actions that the agent takes, which gives positive or negative rewards depending on the goal. Figure 2 illustrates a flowchart over the process, where the agent take actions in an environment that gives a reward and a representation of the state which is returned to the agent.

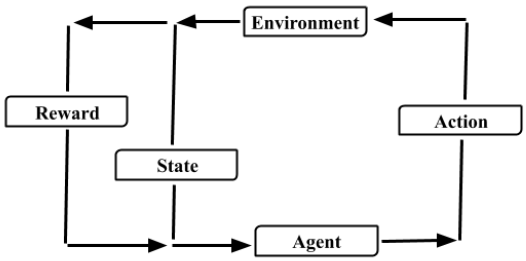

Figure 2. The process over Reinforcement Learning: an agent take actions in an environment that gives a reward and a representation of the state which is returned to the agent.

2.2 Q-learning

One commonly used reinforcement learning method is Q-learning. The method is both active and model-free, meaning that the agent actively selects actions to explore its environment where the reward function is unknown. A data structure called Q-table is used to calculate the maximum expected future reward for actions at each state and will therefore guide the agent to the best action at each state. The Q-table is a matrix with columns representing different actions and rows representing different states. Each cell consists of a Q-value and after each episode, the Q-table and its Q-value is updated. The choice of which states to use depends entirely on the agent's purpose in the application. When the agent chooses the best action based on the associated Q-values, it is called *exploiting*. However, for the agent to be able to know its environment, it must take random actions which is called *exploring*. An important aspect is to save essential data since the learning time increases with the size of the Q-table [1].

Q-learning was developed by Watkins in 1989 [3] and is defined as:

$$Q(s,a) = (1-\alpha)\cdot Q(s,a) + \alpha\cdot(r(s,a) + \gamma\cdot\max_{a\prime} Q(s\prime,a\prime)) \quad (1)$$

Where $Q(s,a)$ is the current Q-value for the state s and action a. The reward value r is obtained from the current action and state. The agent selects a new action a' by referencing to the Q-table with the highest value (max), which results in the agent arriving at the new state s'.

Equation 1 has the parameters *Learning rate* α, which determines to what extent newly acquired information overrides old information and *Discount rate* γ, which balances immediate and future reward. The *Discount rate* is usually a value between 0.8 and 0.99 which is applied to the future reward [4].

2.2.1 Exploring or Exploiting

As mentioned earlier, the agent must explore its environment to allow it to discover new states which otherwise might never have been visited. Therefore, it is important to balance if the agent should explore and make use of the Q-values in the Q-table or exploit the environment and thereby choose random actions. A commonly used method is to implement an Epsilon-greedy function, ε, which controls the behavior of the agent [4]. It is reasonable to assume that the agent from the beginning knows nothing about the environment and thereby the environment should be explored to a greater extent. As the agent explores the environment, the epsilon rate should decrease since the agent learns about the environment and starts to exploit it.

3. Method

The implementation can be divided into two parts, the game set up and the implementation of AI-functionality.

3.1 Game set up

The game was implemented from scratch using *Python* with the library *PyGame*, which is used for making multimedia applications. To create the Q-table, the library *NumPy* was also included, which adds support for multidimensional arrays and matrices. In this game, the player (or AI) was a basket with the goal to catch the falling fruits, which are rectangular objects rendered as images. The quinces are randomly generated at the top of the screen at the start of each episode. There are horizontally ten different channels that the quince can be generated in. In Figure 1, *Score* corresponds to the current score while *Max* is the highest achieved score. Furthermore, the player chooses the speed at which the fruit should fall between two different ones and this is done using a button located in the upper right corner of the screen, see Figure 1. A menu was implemented in order to switch between player mode and AI mode (either training or playing from the Q-table).

3.2 Implementation of AI

In order to implement Q-learning, a number of criteria must be kept in mind. First the actions were defined, which in the project were either to move the basket to the left, right or to do nothing and thus stands still. Furthermore, to update the Q-table, the action-value function Q was implemented from Equation 1 and depends on the actions, states and a variety of parameters. Finally, to gradually reduce the exploration-rate, an epsilon-greedy function was implemented.

3.2.1 States

As mentioned earlier, the aim of the game is for a basket to catch falling fruits. For this to be possible, the most important data to save is the relative distance, horizontally and vertically, from the basket to the quince. This data defines a state which is saved in the Q-table together with the action taken to get there.

3.2.2 Rewards

After each action that is taken, feedback is given to the agent based on events affected by that action. The feedback is given in the form of a reward and is used to update the Q-table using the action-value function Q from Equation 1. In Table 1, the rewards are shown. The agent receives a positive reward for catching the quince (2). In the case when the basket is directly under the quince, a positive reward is also received (1) to inform the agent that it is on the right path, while a negative reward is given if the basket is not directly under the quince (-1). If the agent would fail to catch the quince, a negative reward is given (-100).

Table 1. Rewards given based on a taken action

Event	Reward
Quince over basket	1
Quince not over basket	-1
Catches quince	2
Misses quince	-100

3.2.3 Parameters and the epsilon-greedy function

In Table 2, the parameters used for updating the Q-table with Equation 1 are shown. The exploration rate was initially set to 1 and then slowly decreasing with a factor of 0.99995 for each time step.

Table 2. Parameters used in Equation 1

Parameter	Reward
Learning rate, α	0.85
Discount rate, γ	0.99

4. Result

To determine whether the training gave a good result, the total reward over an episode is summed up and plotted until the agent is trained for almost 2 hours. One episode is the time it takes for a fruit to fall from the top of the screen and down to the agent. Figure 3 shows the plot where the total reward is increasing with the 1600 episodes. The agent can receive a maximum positive reward of 50, this happens if the basket is placed underneath the fruit throughout the episode and then catches it. While, the maximum negative reward is -150 and occurs if the basket is never underneath the fruit nor does it catch it.

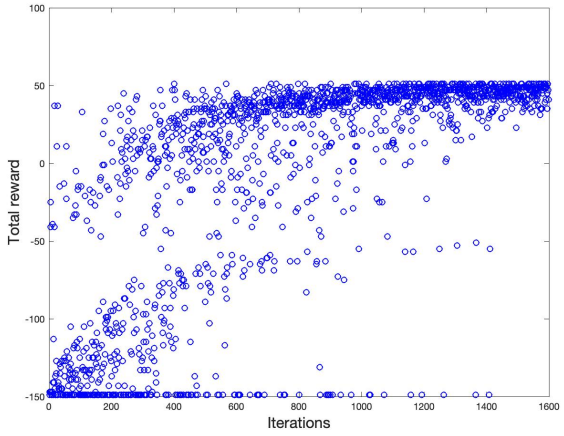

Figure 3. Flowchart over

5. Discussion and Future work

By implementing the game from scratch using *PyGame*, it allowed the game to take form quickly without spending too much time on the graphics. It also helped to maintain full control over the implementation, which in turn provided a higher level of understanding of reinforcement learning and more specifically Q-learning.

In Figure 3, it can be seen that the graph converges towards a stable position with the maximum reward after about 1400 iterations. At the beginning of the graph, very scattered values are displayed. This can be explained by the fact that the agent almost exclusively takes random actions at the beginning as a consequence of the epsilon rate initially set to 1. With increasing episodes, the epsilon rate has decreased by a lot and the agent now almost exclusively takes actions based on the values in the Q-table.

To improve the project, the various parameters could have been investigated and tested with different values. The values used for the learning rate and the discount rate were based on other projects and the underlying theory. This could have been investigated in more detail and thus reduced the learning time. However, due to the time-consuming learning of the agent it was not possible to test different parameters.

Another approach for further development of the project would be to increase the complexity of the game. Either by implementing falling obstacles that the agent should avoid to catch or having multiple fruits falling at the same time during an episode. This would however affect the current implementation of the application since the +1 reward should be given when the basket is under the quince that has come the furthest and thus is closest to the basket. The reward function must therefore be changed to meet this requirement.

6. Conclusion

The aim of this project was to learn an agent given a specific purpose with the reinforcement learning method Q-learning. By observing the result given in Figure 3, the purpose can be considered as accomplished.

References

[1] Ian Millington. *Artificial Intelligence for Games*. Elsevier Inc, 2006.

[2] Andrew G. Barto Richard S. Sutton. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[3] Christopher Watkins. Learning from delayed rewards. 01 1989.

[4] Andre Violante. Simple reinforcement learning: Q-learning. Accessed: 2020-11-04.