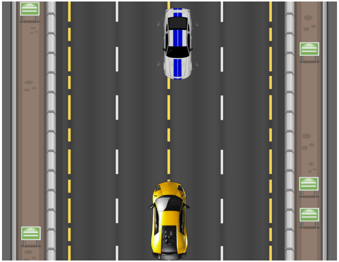
Dodge falling objects game

Davide Rossi

University of Trieste

October 24, 2023



Introduction

What is the game about?

Dodge falling objects is a class of 2D games in which you control a object moving along an axis and you have to dodge other objects moving towards you.

What is the game about?

Dodge falling objects is a class of 2D games in which you control a object moving along an axis and you have to dodge other objects moving towards you.

For each dodged obstacle, you receive one point, the goal of the game is to maximize the number of points.

What is the game about?

Dodge falling objects is a class of 2D games in which you control a object moving along an axis and you have to dodge other objects moving towards you.

For each dodged obstacle, you receive one point, the goal of the game is to maximize the number of points.

Many variants of the game exist with many different interfaces and interactions.

What is the game about?

Dodge falling objects is a class of 2D games in which you control a object moving along an axis and you have to dodge other objects moving towards you.

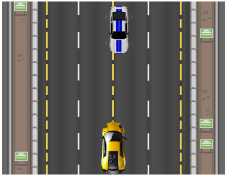
For each dodged obstacle, you receive one point, the goal of the game is to maximize the number of points.

Many variants of the game exist with many different interfaces and interactions.

A simple example is Dodge Mania

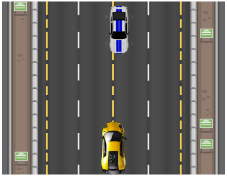
Implementation

For the purposes of the project, a simple implementation has been chosen: you control a car moving along the horizontal axis along a road, you have to dodge another car moving towards you:



Implementation

For the purposes of the project, a simple implementation has been chosen: you control a car moving along the horizontal axis along a road, you have to dodge another car moving towards you:



If your car crashes with the other car or with the side walls you lose:

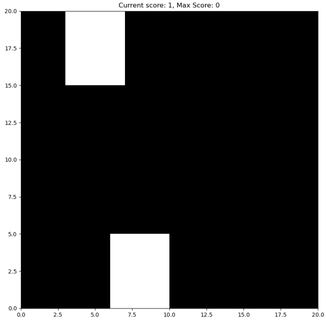
Implementation

The playground is a *height* x *width* grid of 0, with 1 at cars positions.

Implementation

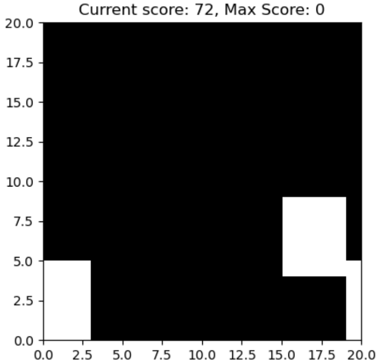
The playground is a *height* x *width* grid of 0, with 1 at cars positions.

Playground class contains the functions to move the cars: at each iteration of the game, enemy car moves one step down, while your car can move one step left or right, or stand still.



Implementation

Also, a simpler variant with a continuous space has been tested. In this case, car cannot crash with the walls, but it will be able to exit from one side and re-enter from the other one:

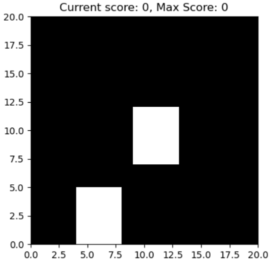


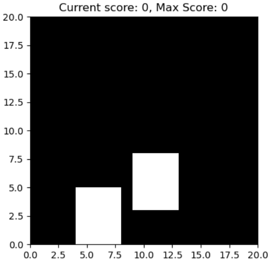
Changing speed

Speed tells how many squares the cars will cross in a single step. It can be manually set, by default it is the same for both cars but some tests have also been done with increasing speed for enemy cars.

Changing speed

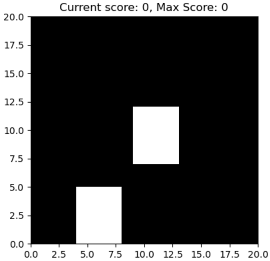
Speed tells how many squares the cars will cross in a single step. It can be manually set, by default it is the same for both cars but some tests have also been done with increasing speed for enemy cars.

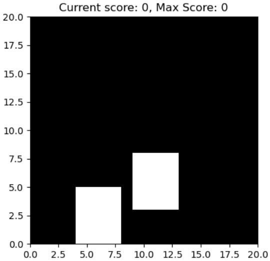




Changing speed

Speed tells how many squares the cars will cross in a single step. It can be manually set, by default it is the same for both cars but some tests have also been done with increasing speed for enemy cars.





To help car to avoid enemies, boost can also be activated: the boosted car will move faster than usual, but this comes with a price...

Reinforcement Learning Framework

Actions Space

There are 5 possible actions:

1. stand still;
2. move right;
3. move left;
4. move right using boost (causes a negative reward);
5. move left using boost (causes a negative reward).

Actions Space

There are 5 possible actions:

1. stand still;
2. move right;
3. move left;
4. move right using boost (causes a negative reward);
5. move left using boost (causes a negative reward).

They have been encoded using a variable `action` with values in `[0,4]`

States Space[1]

To encode the states, state binning has been used to reduce the space size:

- 2 attributes, with values in `[0,3]`, to monitor side obstacles (walls and car) distance;

[1]Note that these are actually observations of the environment, not states

States Space[1]

To encode the states, state binning has been used to reduce the space size:

- 2 attributes, with values in `[0,3]`, to monitor side obstacles (walls and car) distance;
- a boolean to check whether the enemy car is in front of you;

[1] Note that these are actually observations of the environment, not states

States Space[1]

To encode the states, state binning has been used to reduce the space size:

- 2 attributes, with values in `[0,3]`, to monitor side obstacles (walls and car) distance;
- a boolean to check whether the enemy car is in front of you;
- a boolean to check the enemy car position with respect to yours (left or right);

[1] Note that these are actually observations of the environment, not states

States Space[1]

To encode the states, state binning has been used to reduce the space size:

- 2 attributes, with values in `[0,3]`, to monitor side obstacles (walls and car) distance;
- a boolean to check whether the enemy car is in front of you;
- a boolean to check the enemy car position with respect to yours (left or right);
- an attribute with values in `[0,4]` to monitor vertical distance between the two cars

[1] Note that these are actually observations of the environment, not states

States Space[1]

To encode the states, state binning has been used to reduce the space size:

- 2 attributes, with values in `[0,3]`, to monitor side obstacles (walls and car) distance;
- a boolean to check whether the enemy car is in front of you;
- a boolean to check the enemy car position with respect to yours (left or right);
- an attribute with values in `[0,4]` to monitor vertical distance between the two cars

This gives us $4 \times 4 \times 2 \times 2 \times 5 = 320$ possible states.

[1] Note that these are actually observations of the environment, not states

Rewards

Reward is -1000 in case of crash of whatever nature, else a complex reward is assigned at each step, composed by:

- a term which rewards the car for being far from the enemy car

Rewards

Reward is -1000 in case of crash of whatever nature, else a complex reward is assigned at each step, composed by:

- a term which rewards the car for being far from the enemy car
- a term which rewards the car for being far from the walls

Rewards

Reward is -1000 in case of crash of whatever nature, else a complex reward is assigned at each step, composed by:

- a term which rewards the car for being far from the enemy car
- a term which rewards the car for being far from the walls
- -100 if boost has been used
- -1 if car has moved (this encourages the car to stand still)

Solving technique

At each step, we are in a state S, we take an action A and we receive a reward R and a new state S':

```
# get the current state of the environment
state = Agent.get_state(env)

# get the action to be taken based on the state
action = Agent.get_action(state)

# play a step of the game and get the reward and the gameover status
reward, gameover = env.playstep(action, carspeed, policespeed)

# get the new state of the environment after the step
newstate = Agent.get_state(env)

# get the new action to be taken based on the new state
newaction = Agent.get_action(newstate)
```

Solving technique

We'll use Temporal Difference Learning algorithm to estimate and update the state-action value function $Q(s, a)$:

Algorithm 2 TD-Learning

Input Learning Rate $\alpha \in (0; 1]$, small $\epsilon > 0$

Initialize $\hat{Q}(s, a)\ \forall\ s \in S,\ a \in A$

loop for each episode:

  Initialize s

  loop Derive π from $\hat{Q}$ (with $\epsilon - greedy$)

    Choose a from A

    Take A, observe R, S'

    Compute TD-error δ

    Compute eligibility e

    $Q \leftarrow Q + +\alpha\delta e$

    $S \leftarrow S'$

Solving technique

Some notes:

- we have used $TD(0)$, hence the eligibility is the identity function:

$$e = I(S_t = s, A_t = a)$$

Solving technique

Some notes:

- we have used $TD(0)$, hence the eligibility is the identity function:

$$e = I(S_t = s, A_t = a)$$

- The learning rate used for the tests is fixed to $\alpha = 0.01$

Solving technique

Some notes:

- we have used $TD(0)$, hence the eligibility is the identity function:

$$e = I(S_t = s, A_t = a)$$

- The learning rate used for the tests is fixed to $\alpha = 0.01$
- In order to balance exploration and exploitation, an ϵ-greedy policy has been followed, with $\epsilon = 0.5$ decreasing over time

Choosing δ

To compute temporal difference error δ we tested different algorithms:

- SARSA: $\delta = R + \gamma Q(S', A') - Q(S, A)$
- Q-learning: $\delta = R + \gamma \max_{a'} Q(S', a') - Q(S, A)$
- ExpectedSARSA: $\delta = R + \gamma \sum_{a'} \pi(a'|S') Q(S', a') - Q(S, A)$

where γ has been fixed to 1

Results

Training the model

The model has been trained on a 20*20 grid, with variable car size (default 5*4). Each train has been repeated for 5/10 times.

Training the model

The model has been trained on a 20*20 grid, with variable car size (default 5*4). Each train has been repeated for 5/10 times.

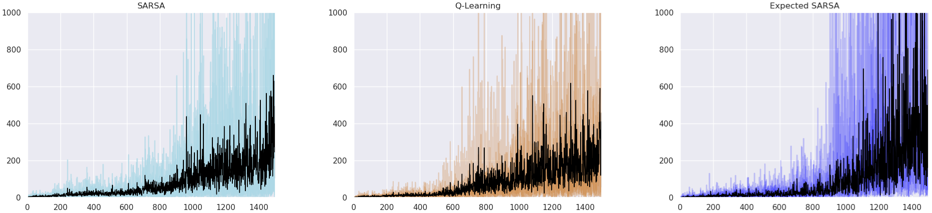
Anyway, learned model is quite flexible and can adapt to different rules (if they are not too restrictive).

Standard environment

Base case: constant speed for both cars

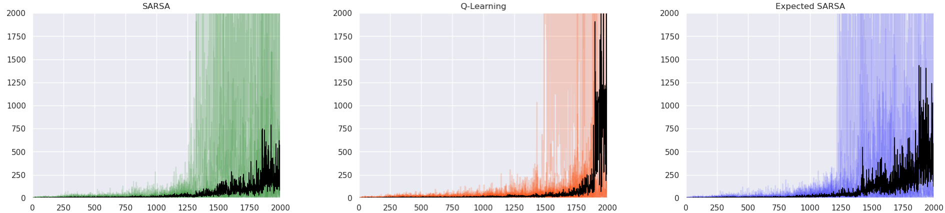
Standard environment

Base case: constant speed for both cars



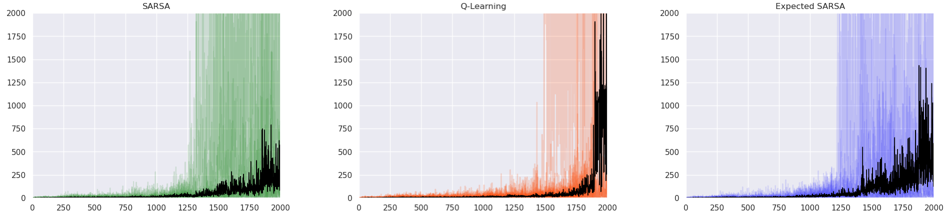
Boost activated

Standard environment

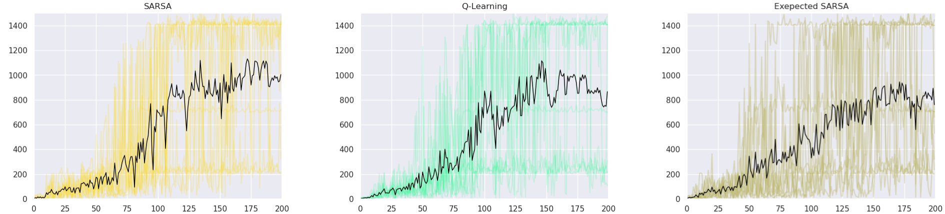


Test with a bigger car (6,5) and boost

Standard environment



Test with a bigger car (6,5) and boost

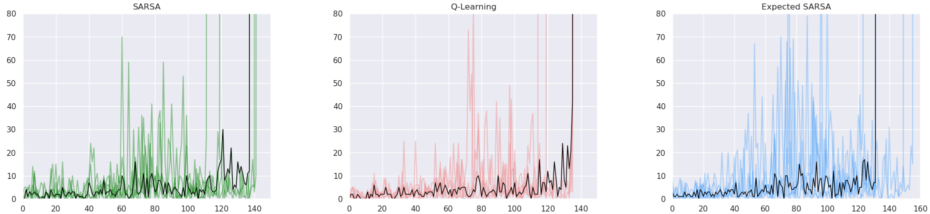


Test with a small car and enemy speed increasing every 100 points

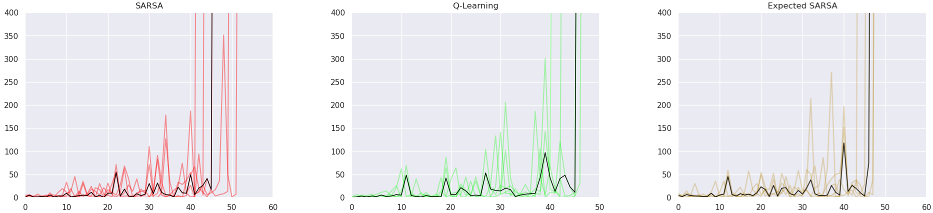
Continuous environment

Base case: constant speed for both cars

Continuous environment

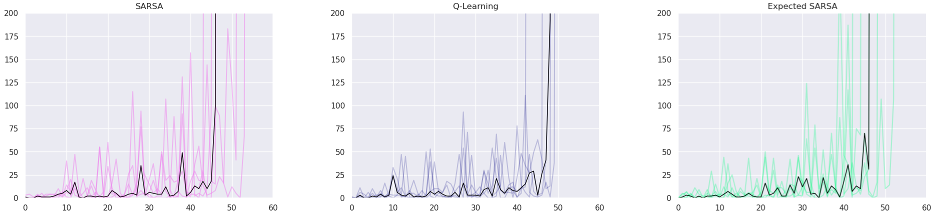


Base case: constant speed for both cars



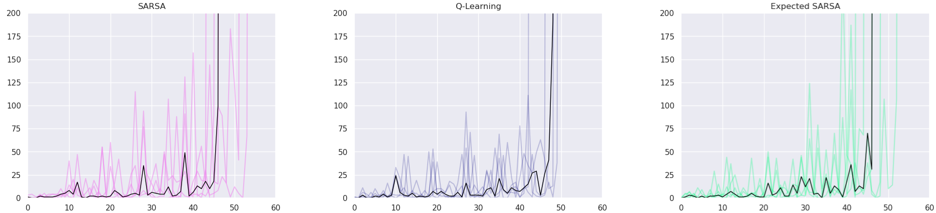
Boost activated

Continuous environment

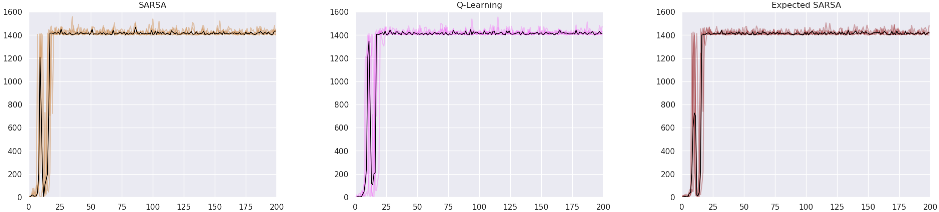


Test with a bigger car (6,5) and boost

Continuous environment



Test with a bigger car (6,5) and boost



Test with a small car and enemy speed increasing every 100 points

Observations

- With Pacman rule, the agent is able to do a perfect score, if rules are not too restrictive (too big car, too fast enemy...)

Observations

- With Pacman rule, the agent is able to do a perfect score, if rules are not too restrictive (too big car, too fast enemy...)
- The learned agent is able to play also with different environment size (if the ratio between car size and environment size doesn't change too much)

Observations

- With Pacman rule, the agent is able to do a perfect score, if rules are not too restrictive (too big car, too fast enemy...)
- The learned agent is able to play also with different environment size (if the ratio between car size and environment size doesn't change too much)
- since we have a online learning, policies can be combined to improve performances

Possible improvements

Parameters

- Non constant learning rate α;

Parameters

- Non constant learning rate α;
- Smaller discount factor: we used $\gamma = 1$, could the program benefit from a smaller value?

Parameters

- Non constant learning rate α;
- Smaller discount factor: we used $\gamma = 1$, could the program benefit from a smaller value?
- Different ϵ decrease

Rewards

- Different penalization in case of crash: smaller, or differentiated according to the type of crash

Rewards

- Different penalization in case of crash: smaller, or differentiated according to the type of crash
- Bigger reward for being far from obstacles

Others

- Test more different environment and car sizes

Others

- Test more different environment and car sizes
- Test more hybrid policies

Others

- Test more different environment and car sizes
- Test more hybrid policies

Some policies regarding these two points are already available in the GitHub repository.

Others

- Test more different environment and car sizes
- Test more hybrid policies

Some policies regarding these two points are already available in the GitHub repository.

- Test different states, particularly in the case of vertical distance

End