24.09.19, Gliwice

Subject:

Biologically Inspired Artificial Intelligence

Project:

Artificial Intelligence learns to play „The World's Hardest Game" with genetic algorithms usage

Authors: Błażej Czaicki
Adam Musiał
Semester: VI
Group: GkiO2
Section: 5
Tutor: dr inż. Grzegorz Baron

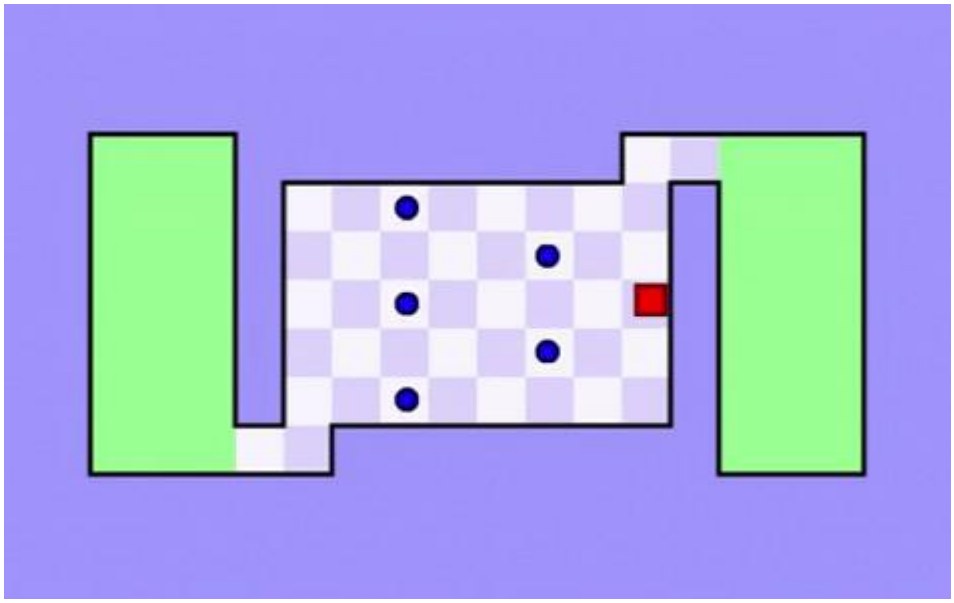
Link to repository:
https://github.com/adammusial777/BIAI-Project/tree/develop

1. Topic

The idea of the project is to create a genetic algorithm and use it to automatically teach the machine (computer) to play the previously written game „The World's Hardest Game".

2. Game Rules And Example

The game consists of moving the red square (player) using arrows and reaching the designated area so as not to touch the obstacles in the form of blue circles. After touching an obstacle, the player is automatically moved to the starting point.



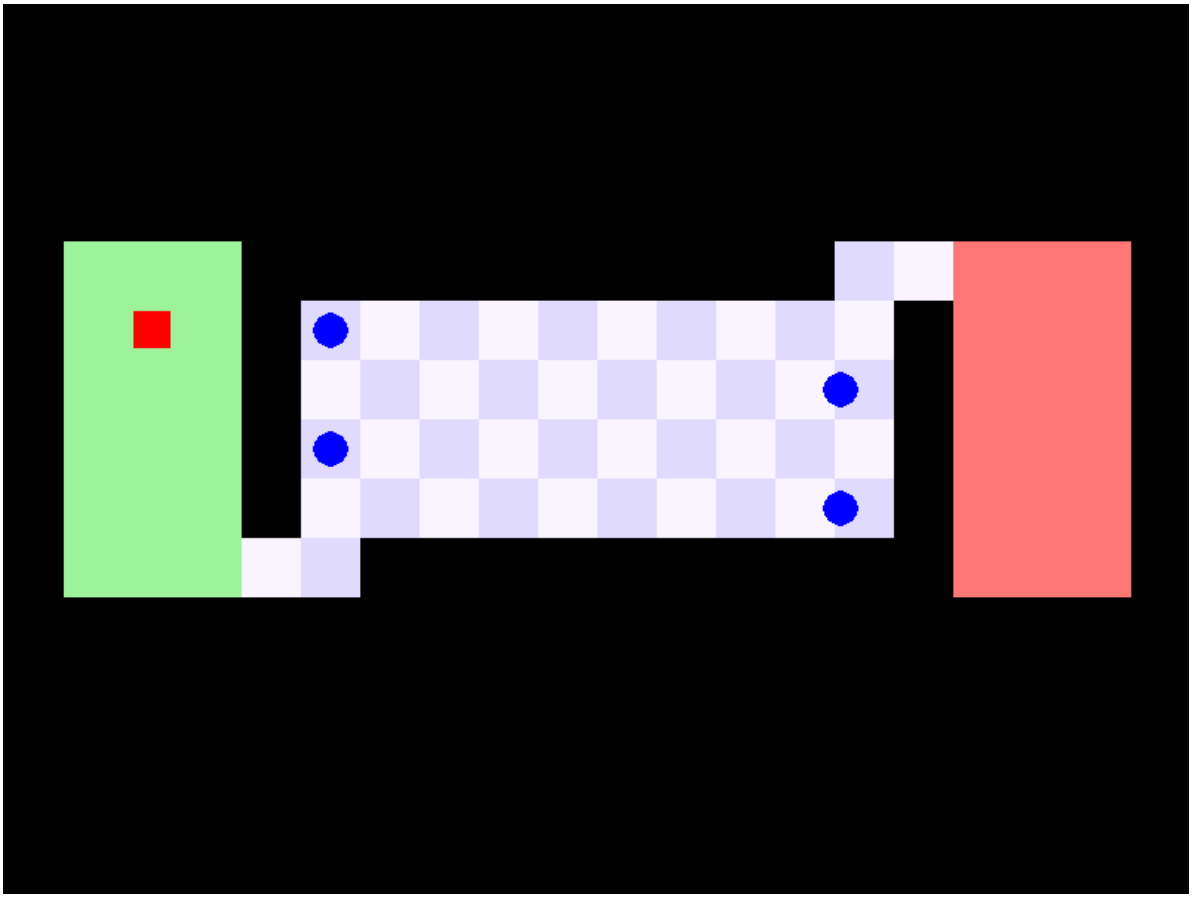
Pic. 1 *„The World's Hardest Game" example*

3. Genetic Algorithm

Genetic algorithm contains set of operations on chromosomes. It calculate fitness on which selection based. Afterwards selection, part of genes is mutating due to diversity is maintained. This is how new generation is created.

4. Game Implementation

Initially, a game was implemented that could have been played by a human being. The control was carried out by means of arrows.



Pic. 2 *Our own „The World's Hardest Game"*

5. Artificial Intelligence Implementation

GeneticAlghoritm class Begin from computing Fitness function for every chromosome. The method calculate fitness based on distance between player and current target. Reaching the area is rewarded. The next step is sorting list of chromosomes from the smallest to the biggest fitness and calculating sum of fitnesses. Selection method choose chromosomes to new list from better part of list of chromosomes. In a further step some chromosome's genes are mutating. New generation are increasing about specified number of genes unless exceed a limit. At the end of the day some of obsolete variables are reset.

6. Internal Specification

The following classes are used in the game:

- **BoardReader** - is responsible for loading the board from the file.
- **Collider** - is responsible for checking if a collision has occurred.
- **DeltaTime** - is responsible for the correct frame rate of the game.
- **Enemy** - is responsible for graphic and program representation of obstacles in the form of blue circles and their movement.
- **GameObject** - is responsible for storing color and coordinates. The following classes inherit from it: Enemy, Player, Tile and TargetArea .
- **Player** - is responsible for representing a manually controlled player, updating, moving, solving collisions, checking position and his respawn. PlayerComputer inherit from it.
- **PlayerComputer** - is responsible for the representation of a player created for learning by a genetic algorithm, its update, movement and death.
- **Chromosome** - is responsible for representing a single chromosome, updating it, drawing the direction of movement, initializing, mutating and increasing the number of genes, as well as calculating fitness.
- **Genetic Algorithm** - responsible for the course of updating, counting the sum of fitness, selecting the best individuals, passing the best genes to the next generation and chromosome mutation.
- **TargetArea** - is responsible for graphic and software representation of checkpoints, calculating the distance between him and the player, and checking whether it has been achieved.
- **ResultsSaver** - is responsible for saving the fitness, distance covered and the time elapsed between the launch of the programme and the death of a generation to *. csv file.
- **Tile** - is responsible for representation of a single tile of board.
- **GameManager** - is responsible for creating and managing objects of the above mentioned classes and includes the main game loop.

7. Testing/Changes In Code

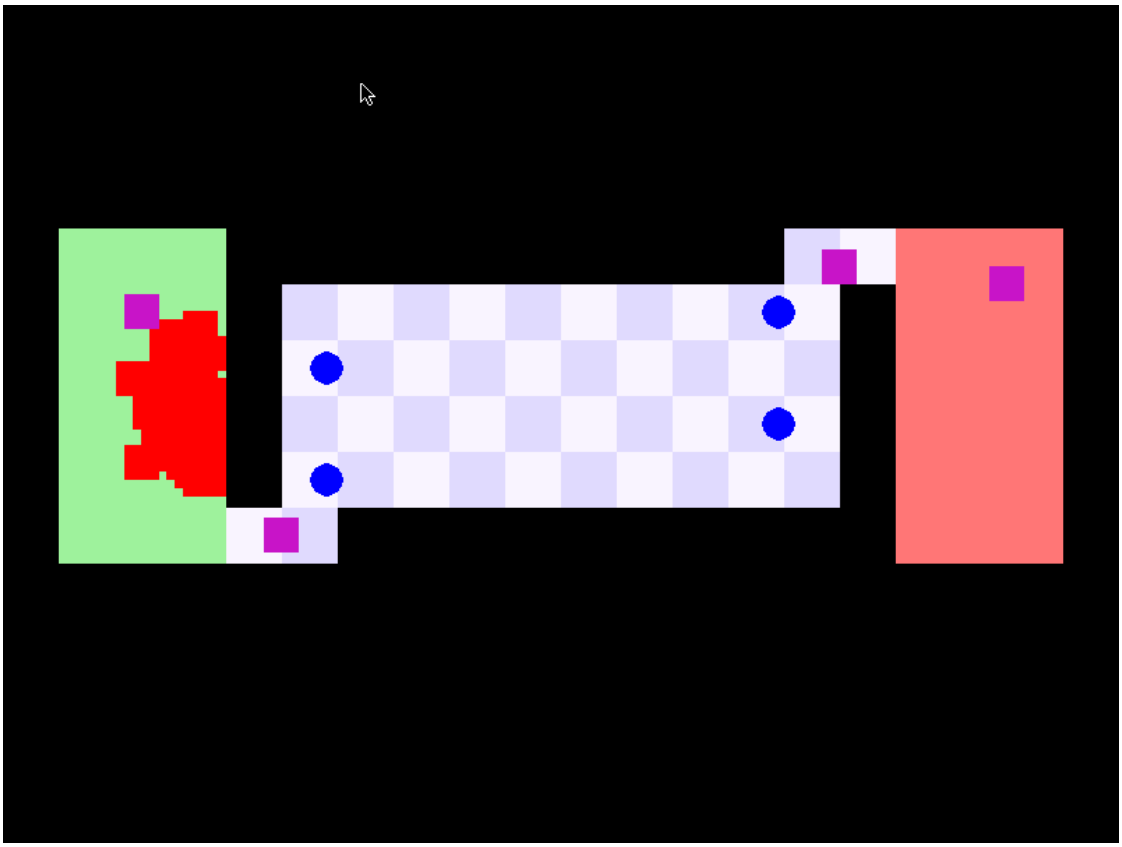
While testing the program, we noticed that it has a problem with detecting collisions with obstacles, so the artificial intelligence learning stopped at the first contact with them. Automatically generated players did not avoid obstacles, but either died on them or after a long time turned back in the direction of the starting point.

We performed tests for an initial number of 50 genes, iterating every 10 genes up to a maximum of 300 genes. We tried to increase the maximum achievable number of genes to 600 so that the algorithm would have more time and possible movements to learn. However, this has not had any effect.

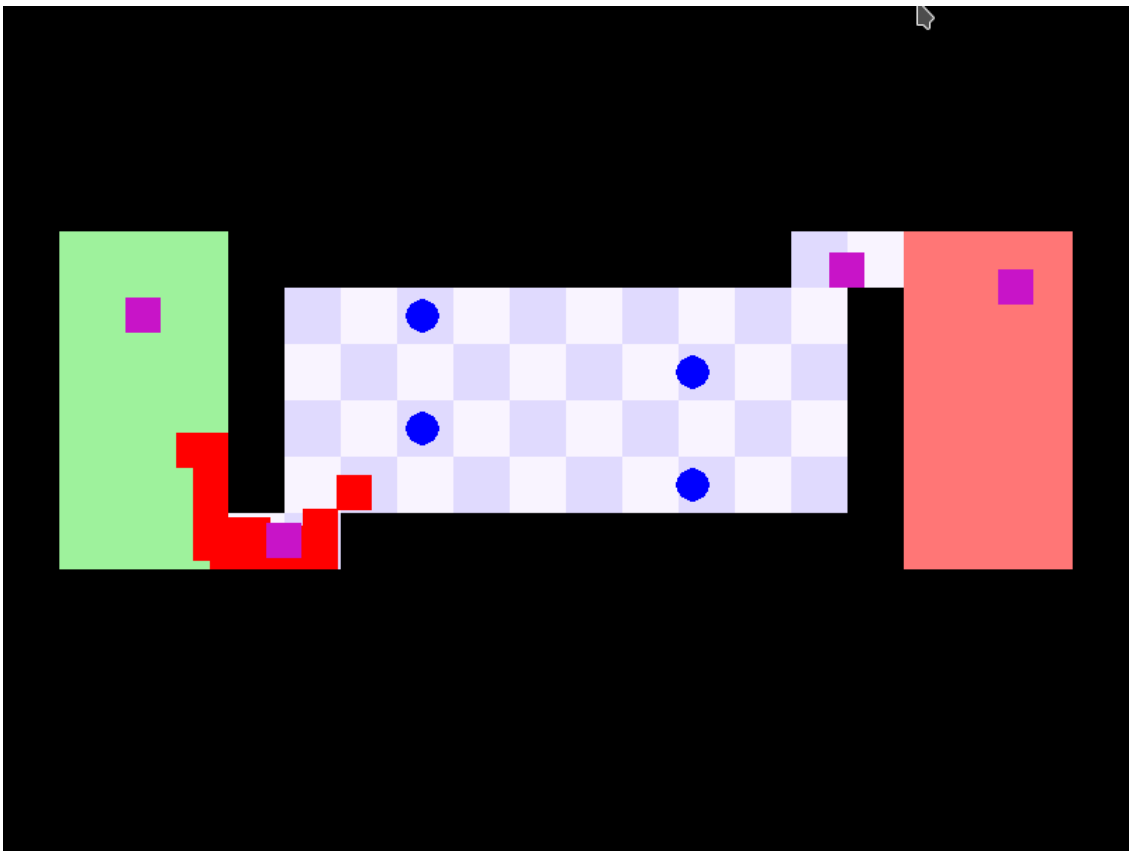
We decided to turn off the system for detecting collisions with obstacles and check how learing goes without them. The effect was positive, the genetic algorithm after evolution over the next generations fulfilled its role and automatically generated players reached their destination.

8. Results

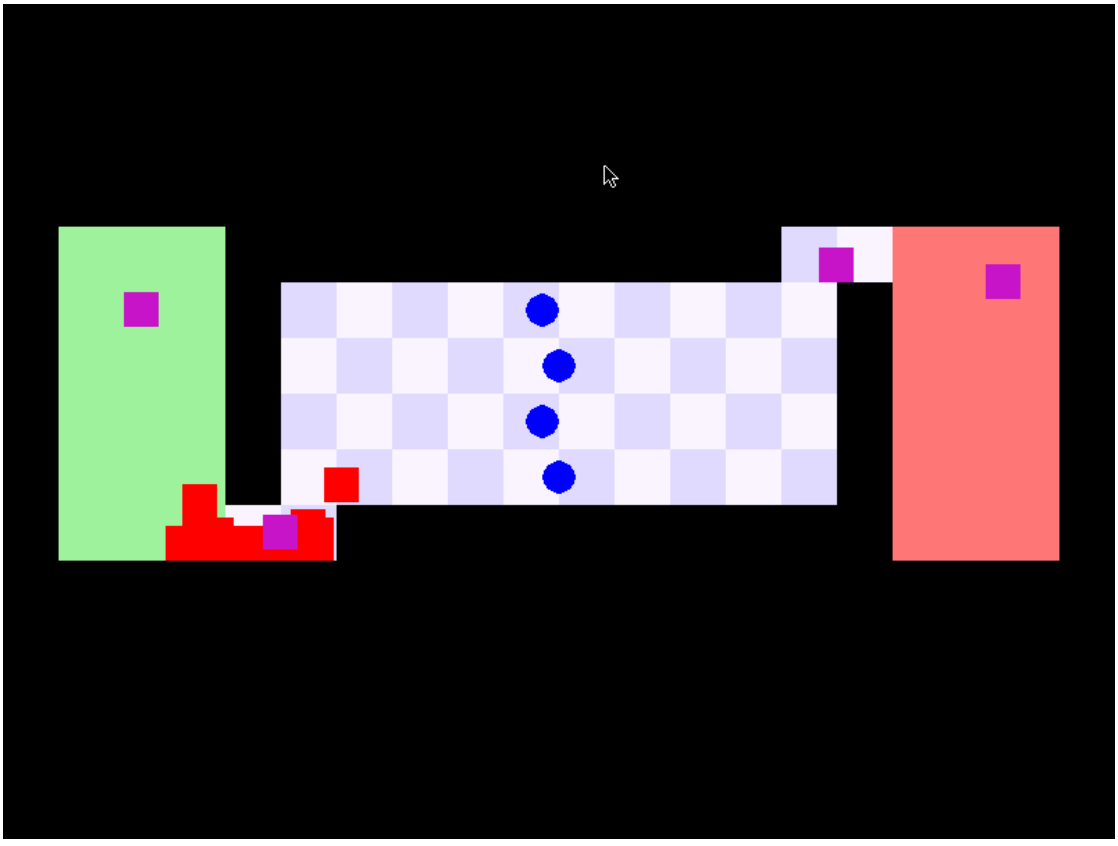
Below, in the form of screenshots, is shown the process of learning the subsequent generations.



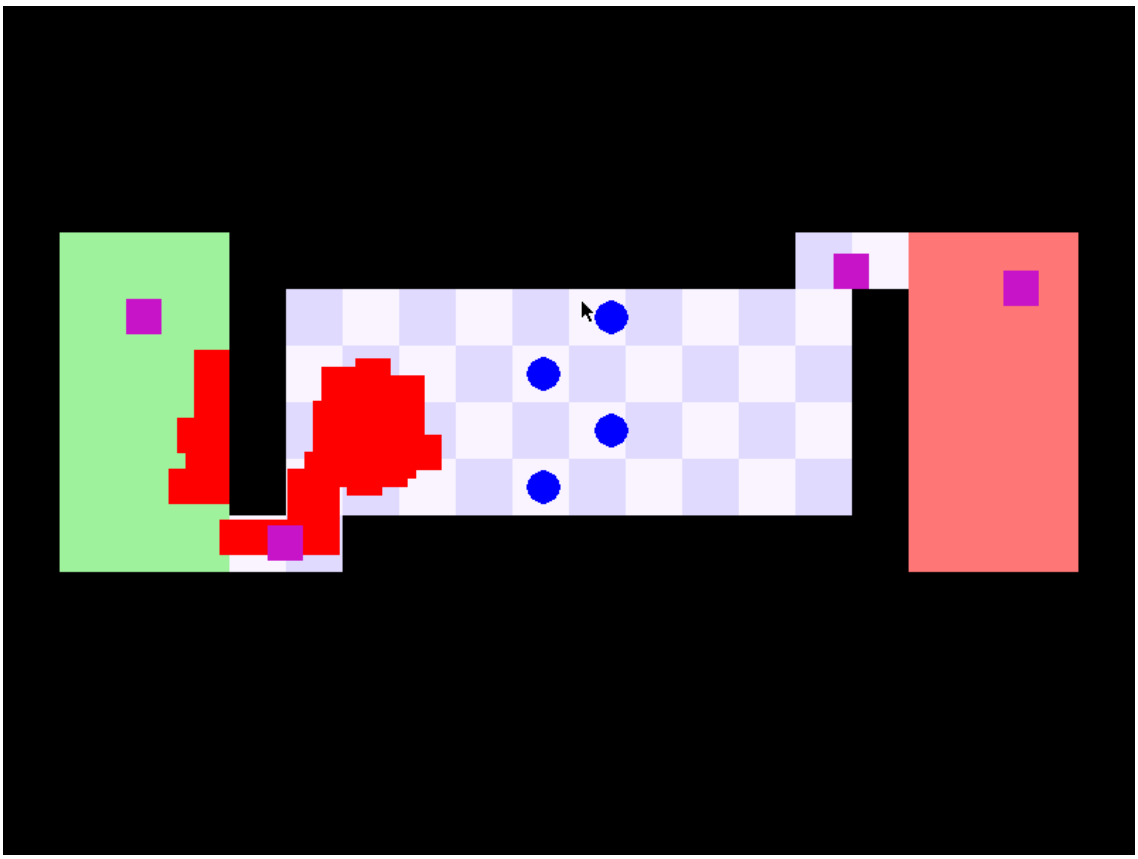
Pic. 3 *The process of learning the subsequent generations (1)*



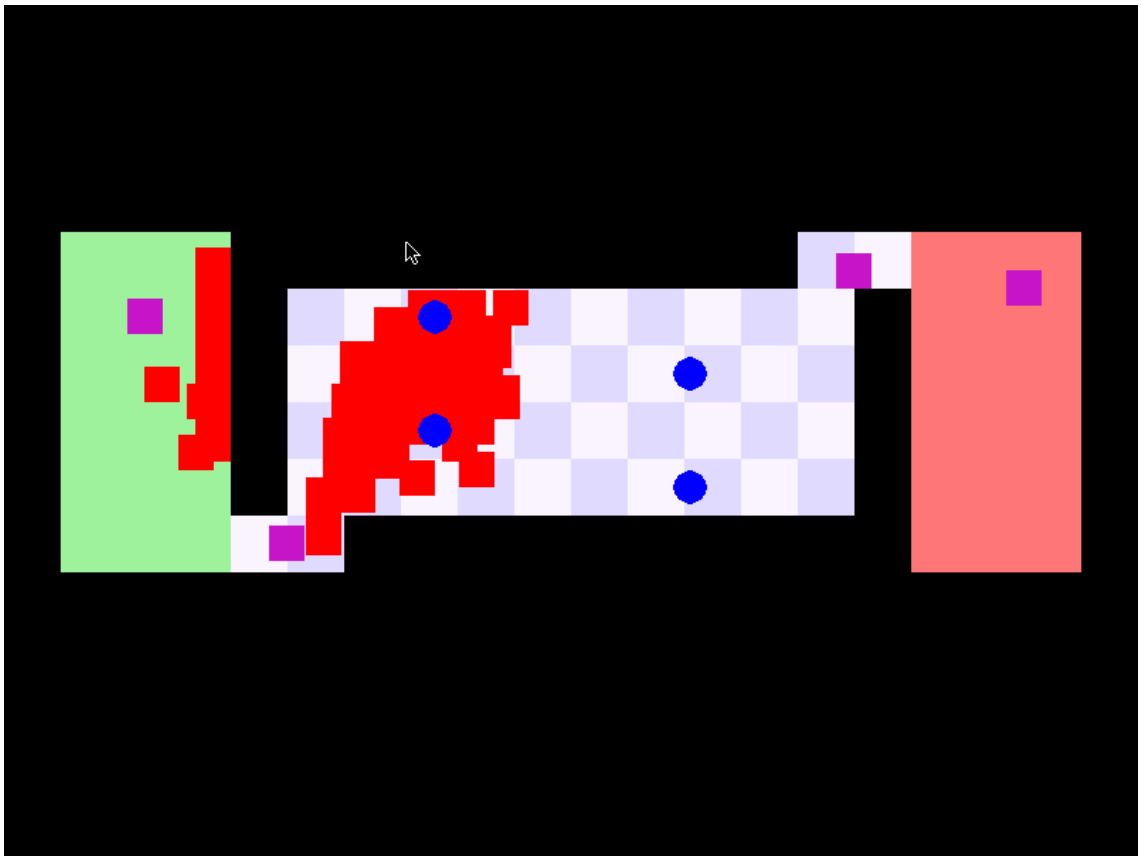
Pic. 4 *The process of learning the subsequent generations (2)*



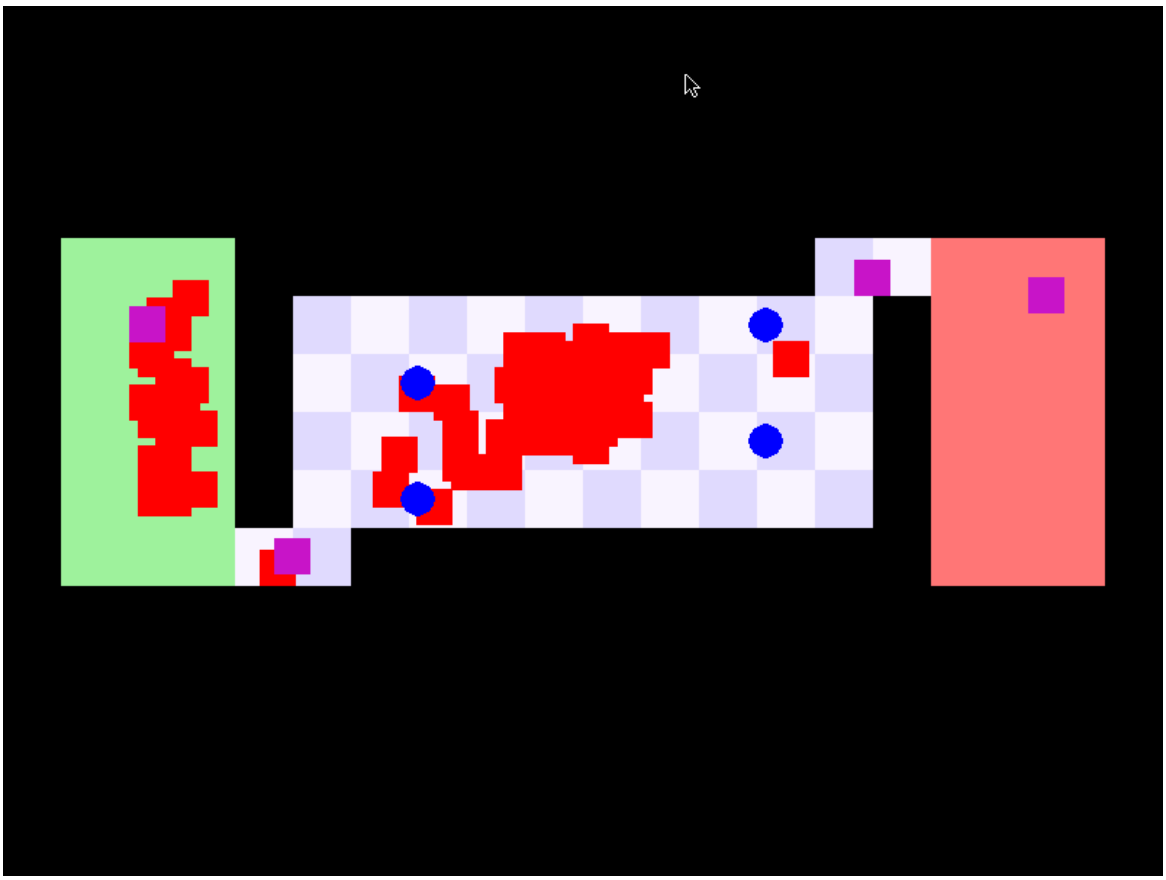
Pic. 5 *The process of learning the subsequent generations (3)*



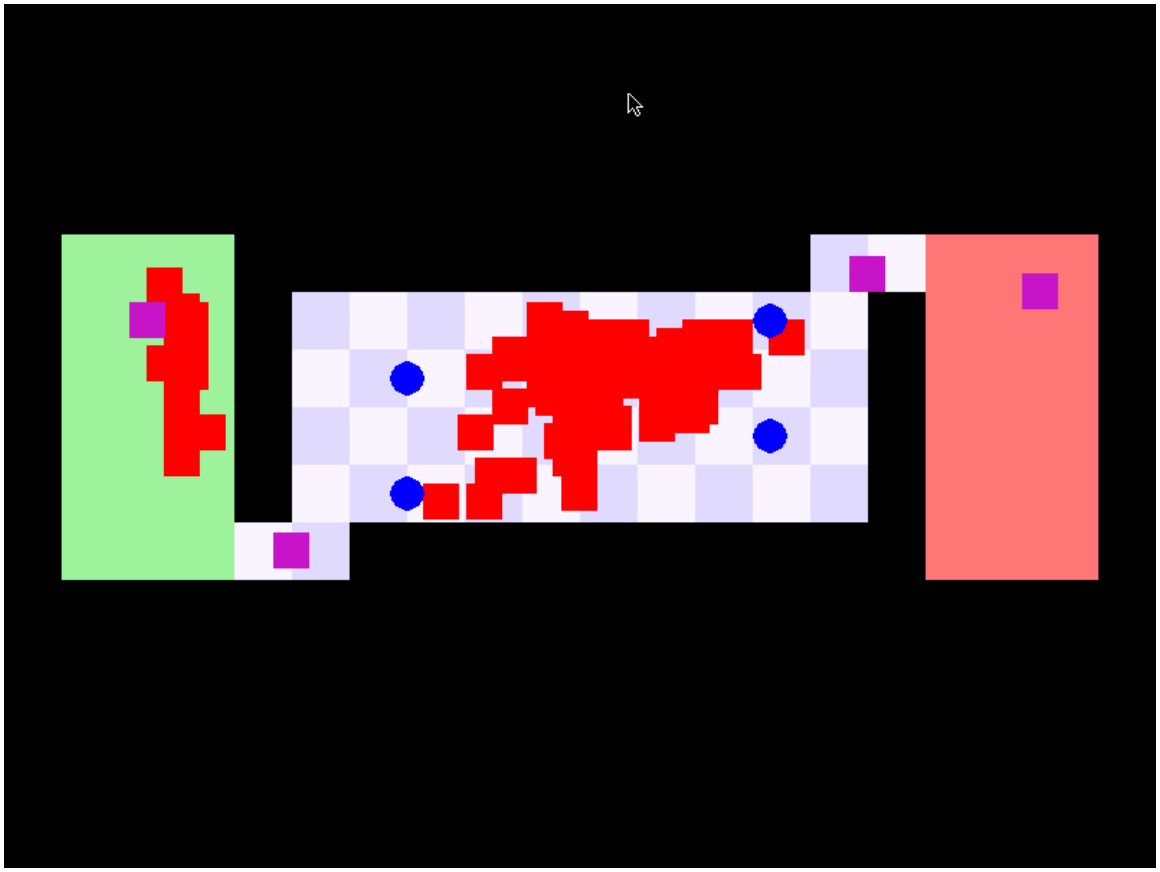
Pic. 6 *The process of learning the subsequent generations (4)*



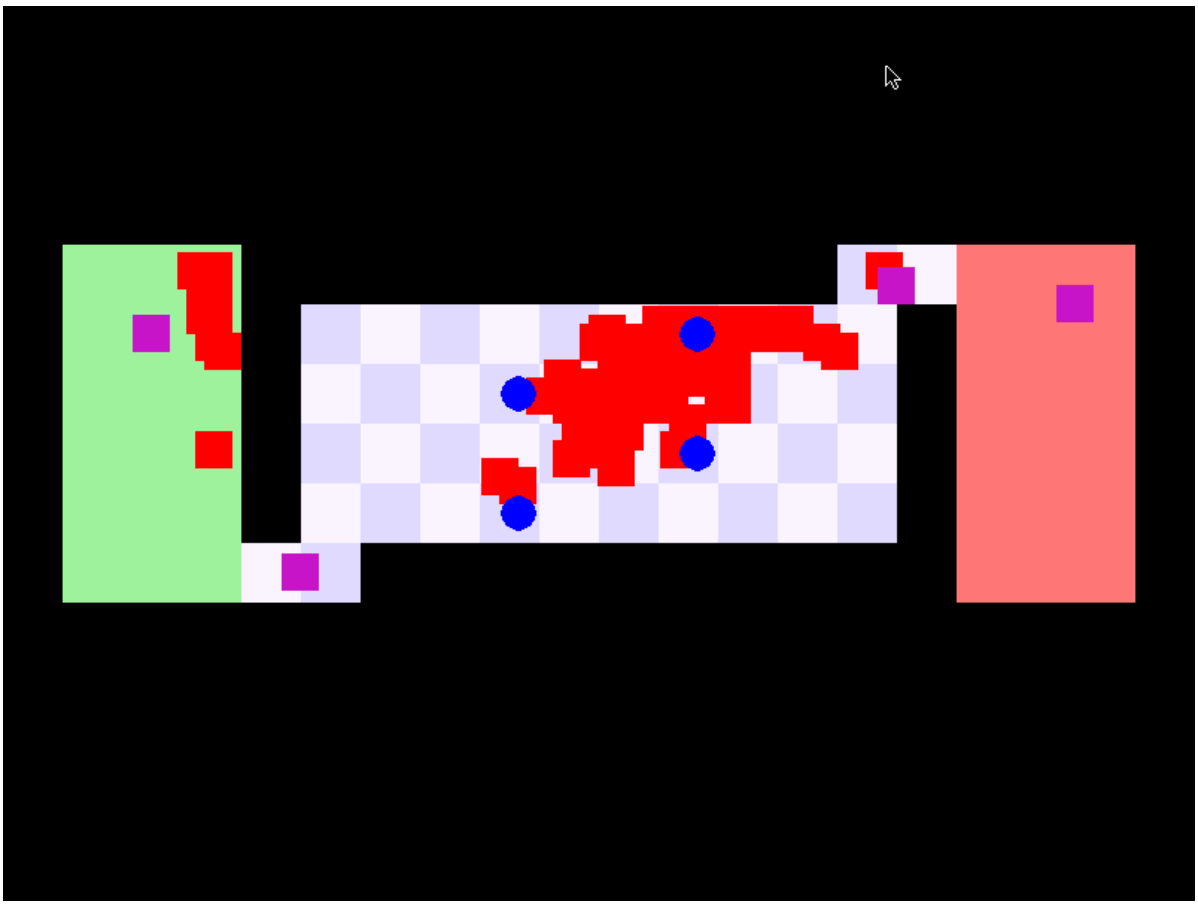
Pic. 7 *The process of learning the subsequent generations (5)*



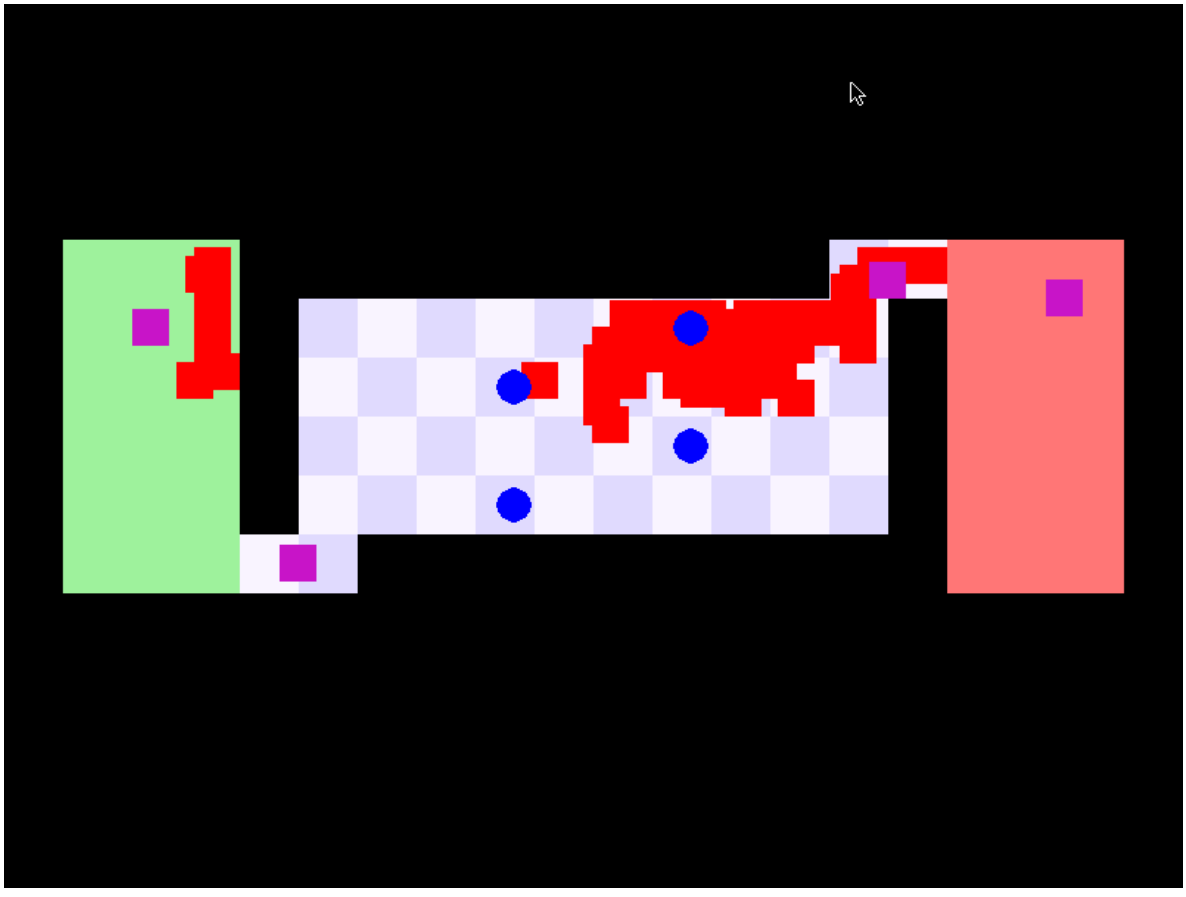
Pic. 8 *The process of learning the subsequent generations (6)*



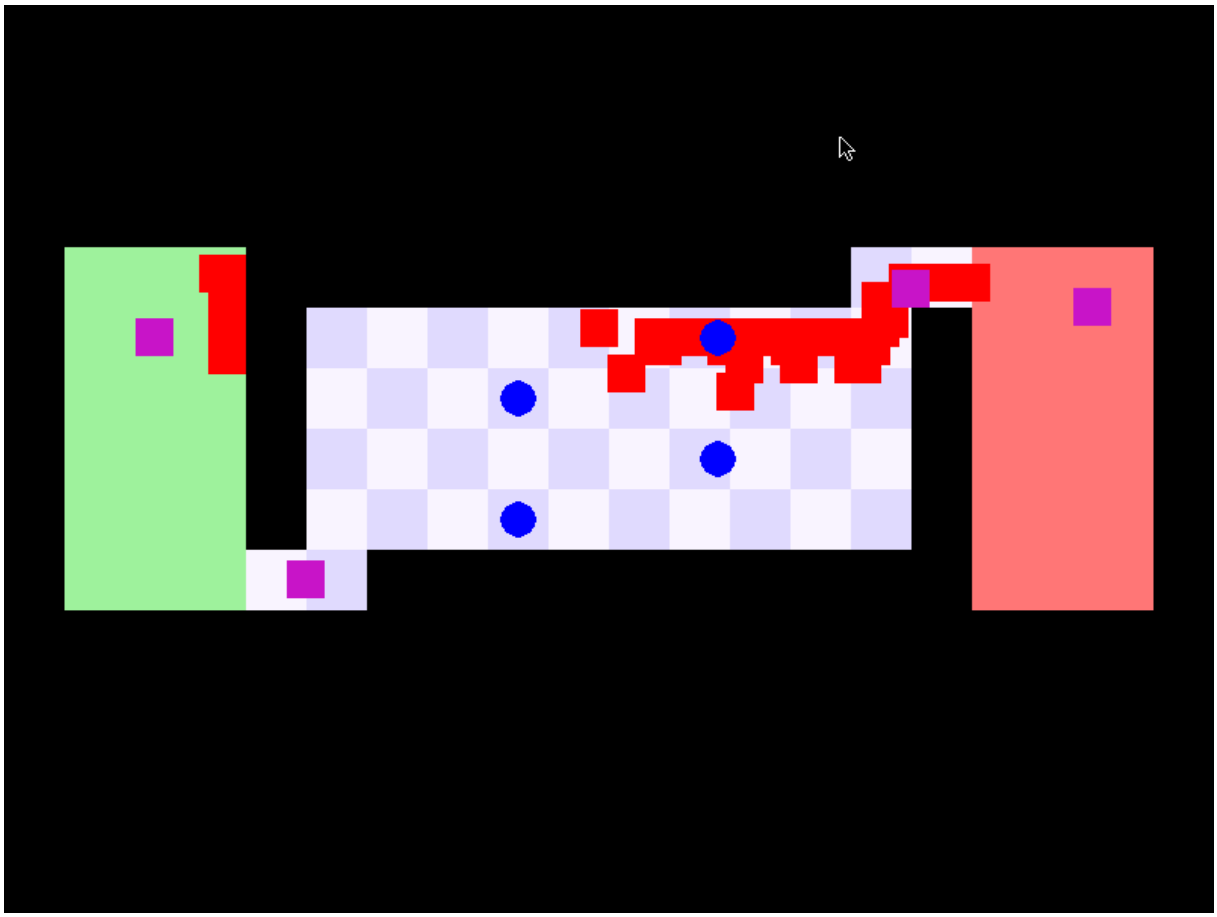
Pic. 9 *The process of learning the subsequent generations (7)*



Pic. 10 *The process of learning the subsequent generations (8)*

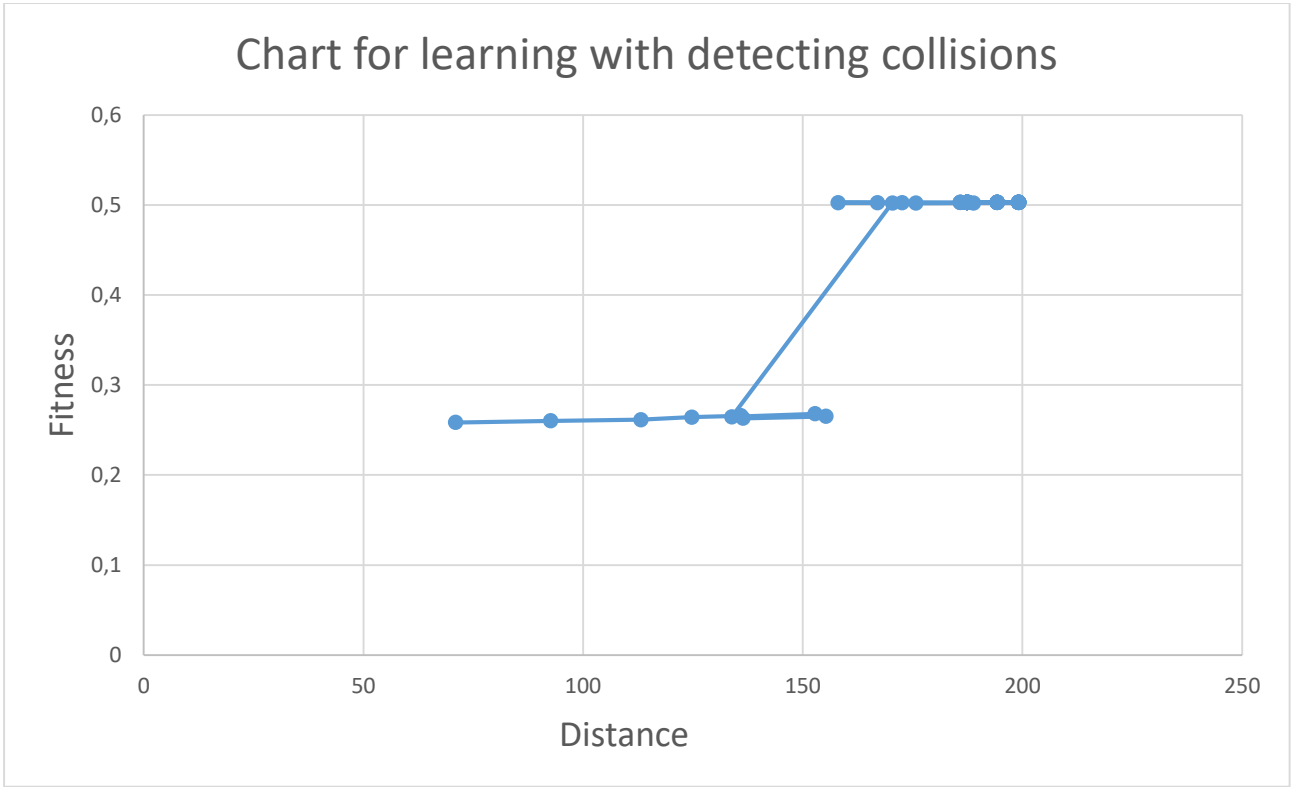


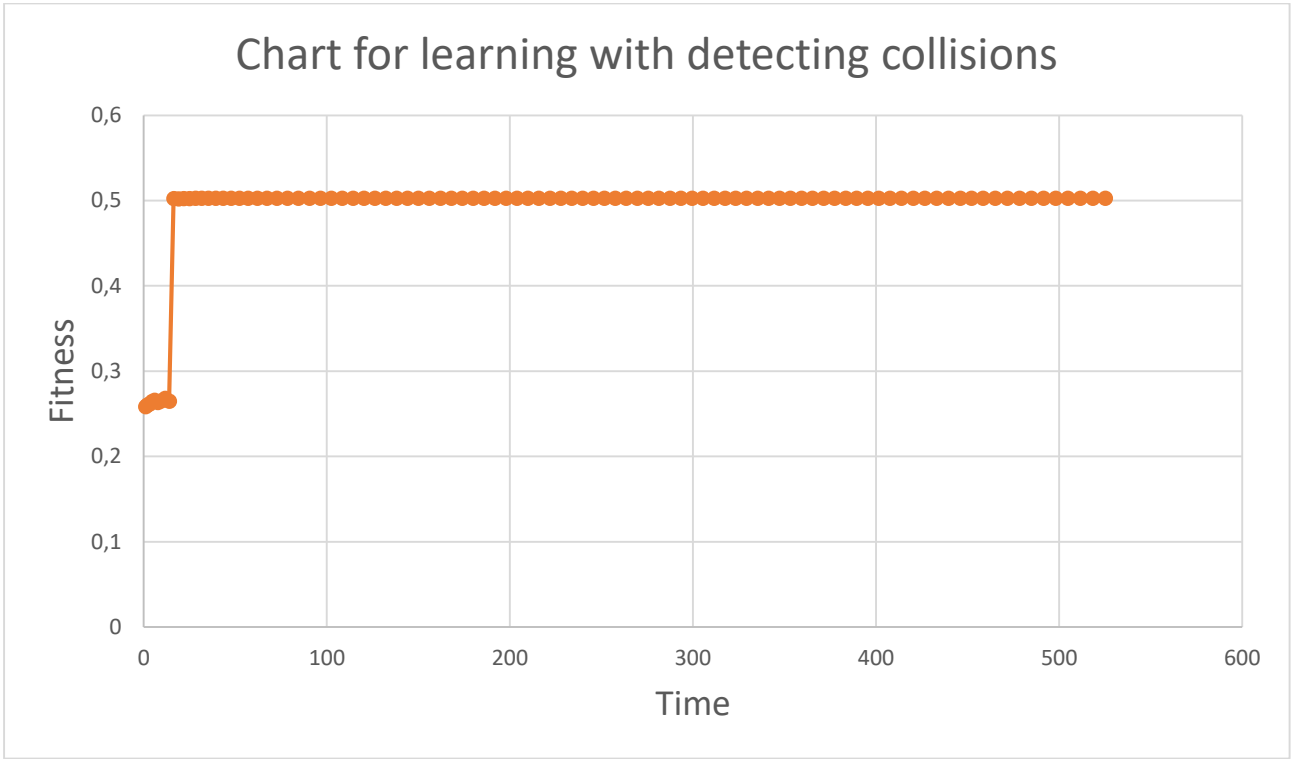
Pic. 11 *The process of learning the subsequent generations (9)*



Pic. 12 *The process of learning the subsequent generations (10)*

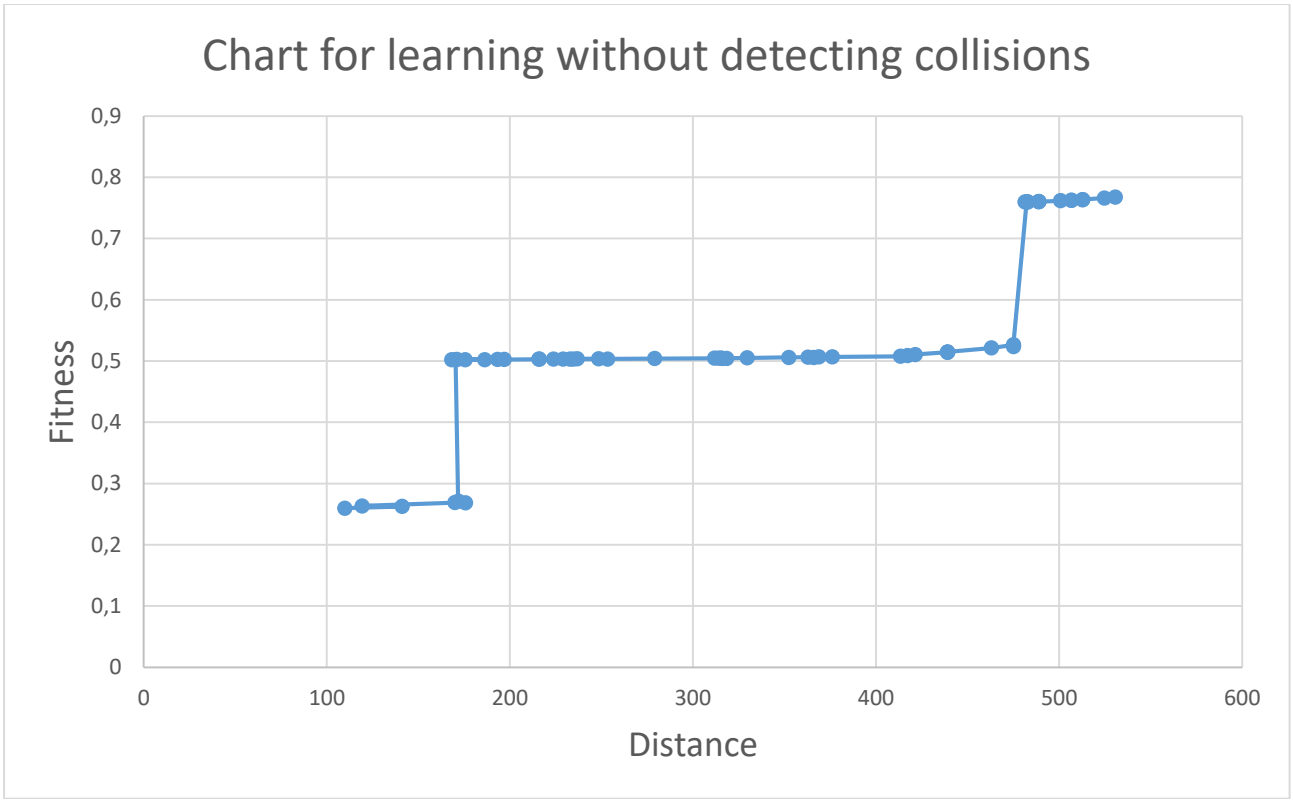
The following charts show the dependence of fitness on distance and time for learning with detecting collisions:

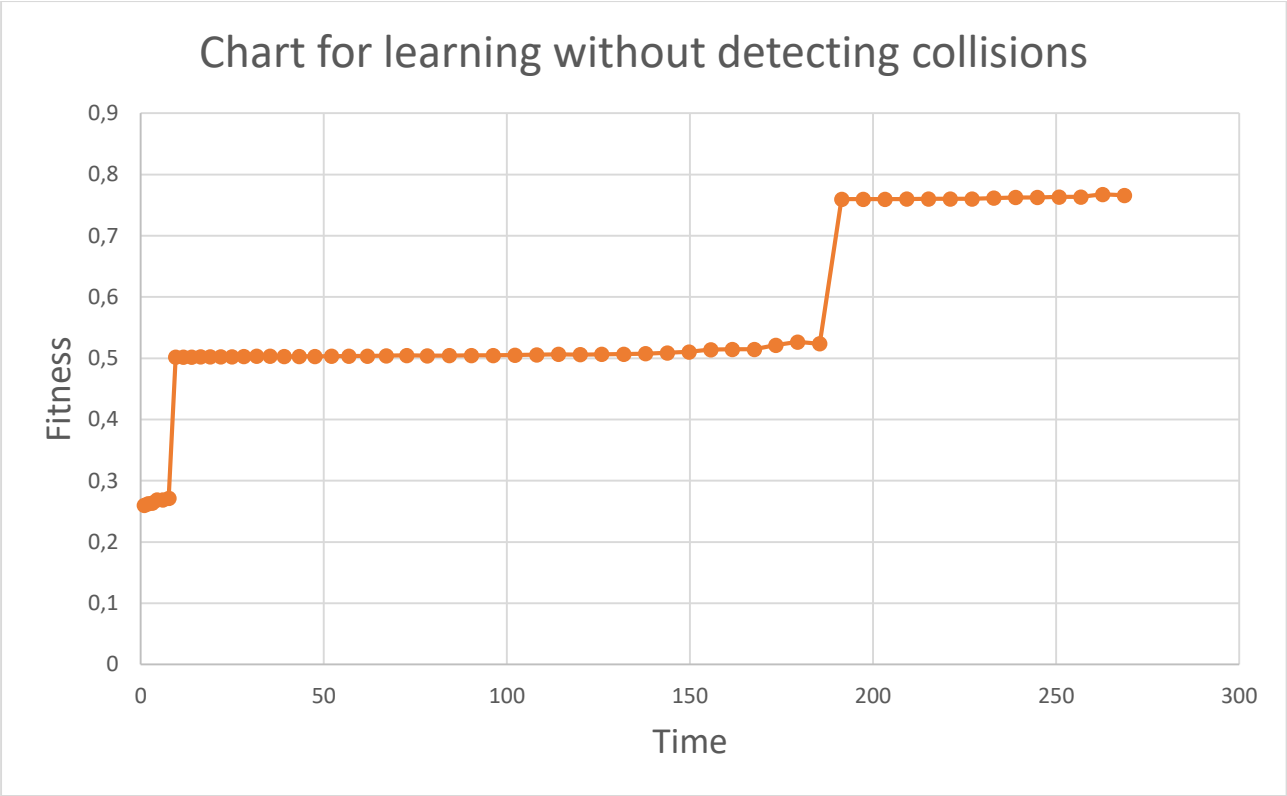




As

The following charts show the dependence of fitness on distance and time for learning without detecting collisions:





As you can see on the first two charts, the fitness stopped at ~ 0,5 and did not increase further due to the inability to overcome obstacles, while on the next two charts it reached ~ 0,75 and reached the end point.

9. Conclusions

Creating a properly functioning genetic algorithm was a real challenge. We have not managed to achieve satisfactory results in terms of avoiding obstacles by artificial intelligence, but we have learned how to create a genetic algorithm and how much it reflects biology and evolution. The Python programming language itself was something completely new for us in this project and solving many of the problems involved was time-consuming.

10. Sources

- https://www.pygame.org/docs/
- https://docs.python.org/3/tutorial/index.html
- https://www.youtube.com/watch?v=Yo2SepcNyw4&t=
- https://code-bullet.github.io/WorldsHardestGameAI/WHG/
- https://github.com/Code-Bullet/WorldsHardestGameAI