3D Minigolf

An Analysis of Different Approaches handle complex maze-like courses

Project 1-2
Phase 3
Coordinator : Carlo Galuzzi

Samuel Kopp
Siemen Geurts
Hendrik Baacke
Ruben Bijl
Annelot Pruijn

Maastricht University

Problem definition

- Extend AI bot for maze-like courses
- Introduce small random error, which the AI bot should deal with
- Hold a competition between the bots

Contents

- Physics Engine
- Terrain Generation
- Search Algorithms
- Experiments
- Conclusion

Physics Engine

Runge-Kutta 4^{th} order:

$$k_{i,1} = h_i \cdot f(t_i, w_i)$$

$$k_{i,2} = h_i \cdot f\left(t_i + \frac{1}{2}h_i, w_i + \frac{1}{2}k_{i,1}\right)$$

$$k_{i,3} = h_i \cdot f\left(t_i + \frac{1}{2}h_i, w_i + \frac{1}{2}k_{i,2}\right)$$

$$k_{i,4} = h_i \cdot f\left(t_i + h_i, w_i + k_{i,3}\right)$$

$$w_{i+1} = w_i + \frac{1}{6} \cdot (k_{i,1} + 2k_{i,2} + 2k_{i,3} + k_{i,4})$$

t_i = time at step i

w_i = function approximation at t_i

h_i = the timestep, i.e. t_{i+1} - t_i

f = the derivative function of the desired function

Runge-Kutta 4th order vs Euler

Euler's Method (1^{st} order):

$$w_{i+1} = w_i + h \cdot f(t_i, w_i)$$

- How do they compare in computational time?
  - 1 iteration (i.e. updating the position of the ball once):
    - Euler's method : 336 885 ns
    - RK4: 1 414 861 ns
- Is this an issue?
  - 1/60th of a second: 16 666 666 ns

Physics engine

Euler vs. Runge-Kutta 4th order: Euler / Runge-Kutta

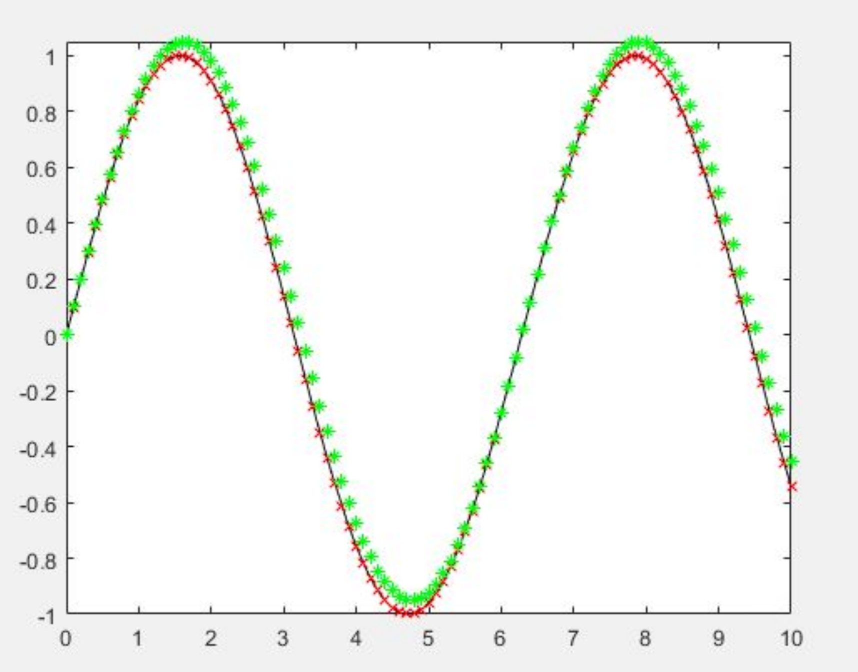


fig 1

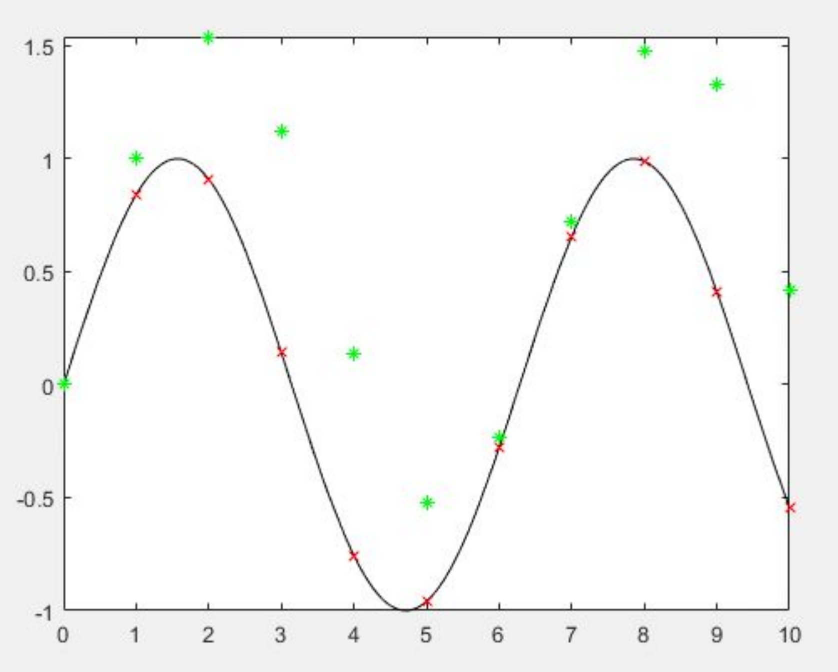
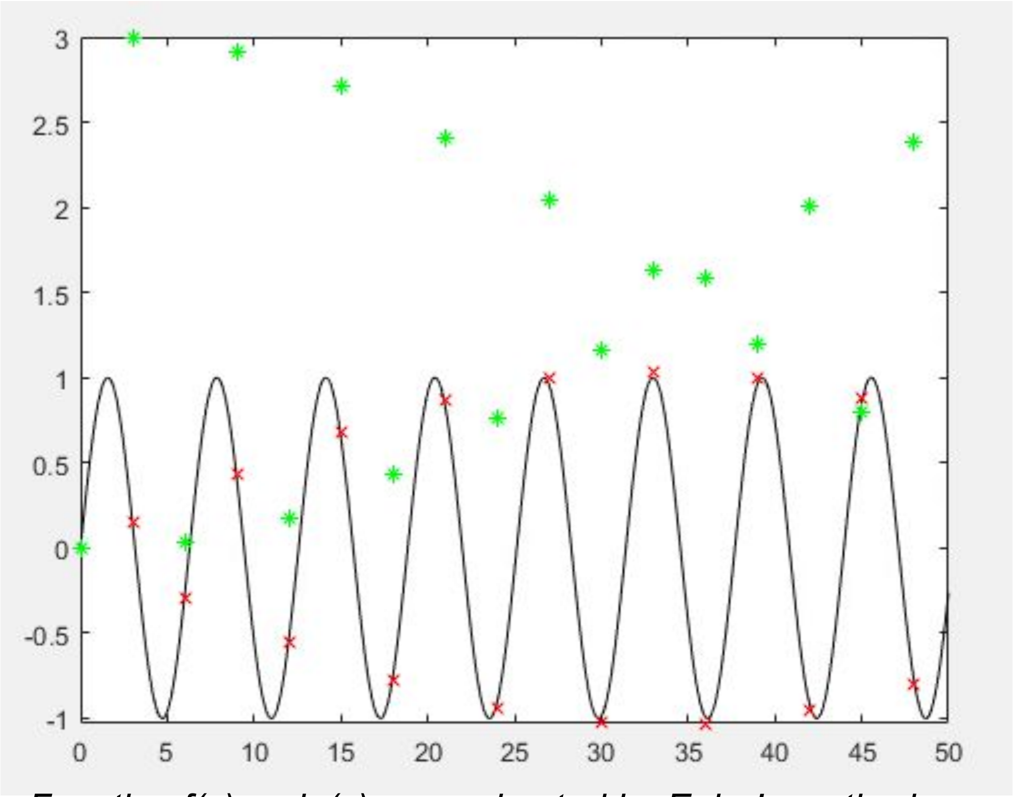


fig 2

Function f(x) = sin(x) approximated by Euler's method and Runge-Kutta 4th order method with stepsize 0.1 (fig 1) and stepsize 1 (fig 2)

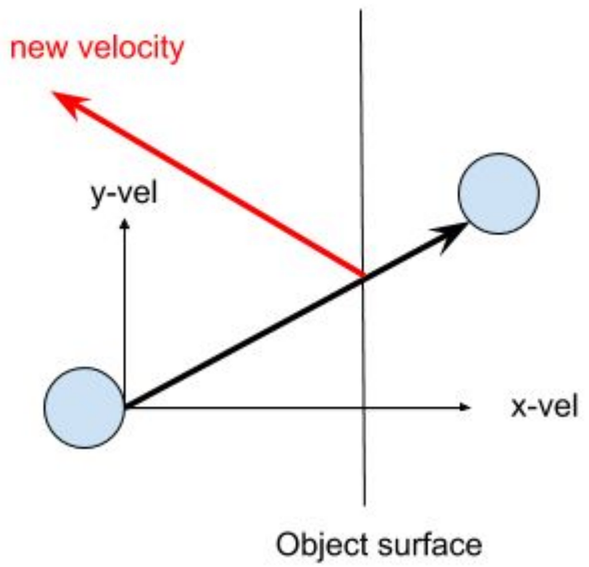
Runge-Kutta 4th order vs Euler

- Euler is faster
- Runge-Kutta is more accurate and reliable
- Runge-Kutta doesn't sacrifice computational time while remaining accurate



Function f(x) = sin(x) approximated by Euler's method and Runge-Kutta 4th order method with stepsize 3

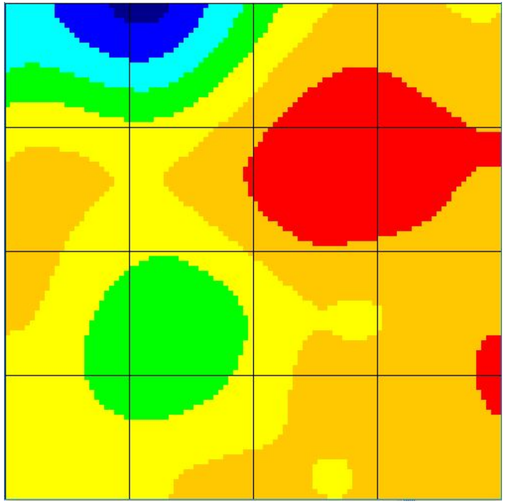
Collision detection

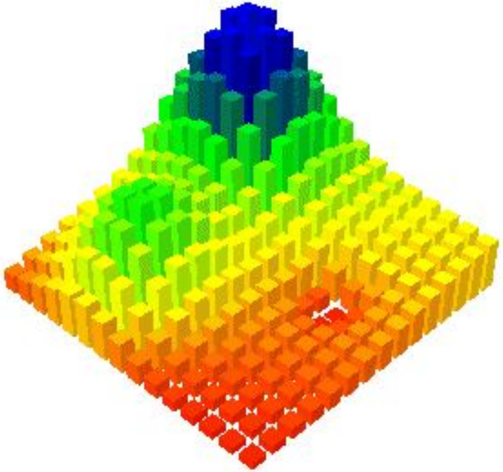


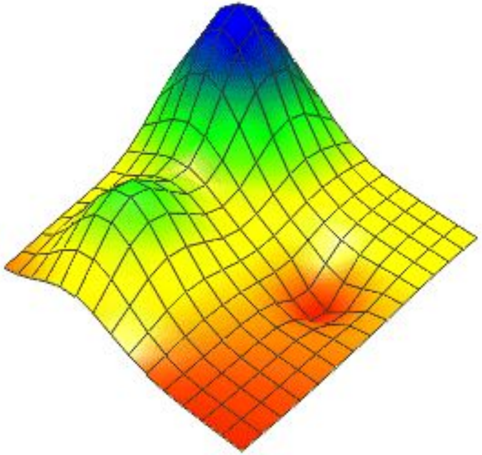
- Check coordinates
- Find intersection of velocity and object surface
- Reflect back accordingly

Terrain Generation

- Set of points
- Bicubic Interpolation
- Regular Grid of Unit Squares
- Polynomials describing surface



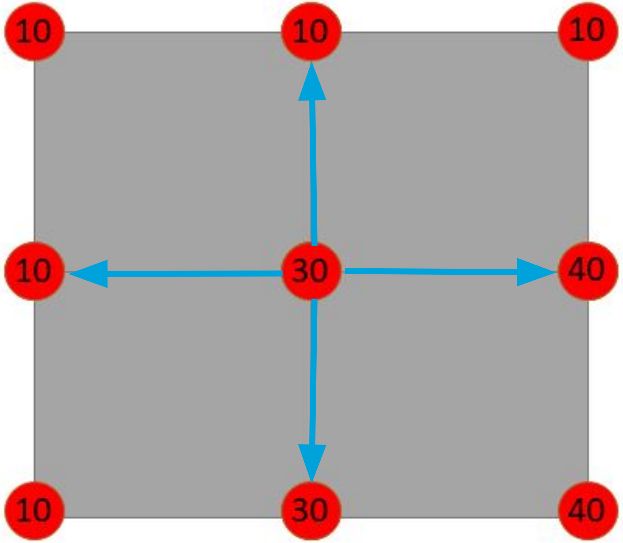




Terrain Generation

- Polynomials describing surface
- Sets of coefficients *a* → SLE

$$p(0,0) =$$



$$(1) \quad p(x,y) = \sum_{i=0}^{3} \textstyle\sum_{j=0}^{3} a_{ij} x^i y^j$$

$$(2) \quad p_x(x,y) = \sum_{i=1}^{3} \sum_{j=0}^{3} a_{ij} i x^{i-1} y^j$$

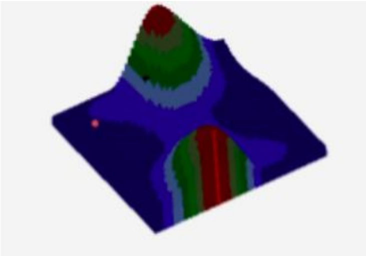
$$(3) \quad p_y(x,y) = \sum_{i=0}^{3} \sum_{j=1}^{3} a_{ij} x^i j y^{j-1}$$

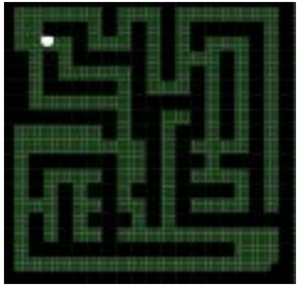
$$(4) \quad p_{xy}(x,y) = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} i x^{i-1} j y^{j-1}$$

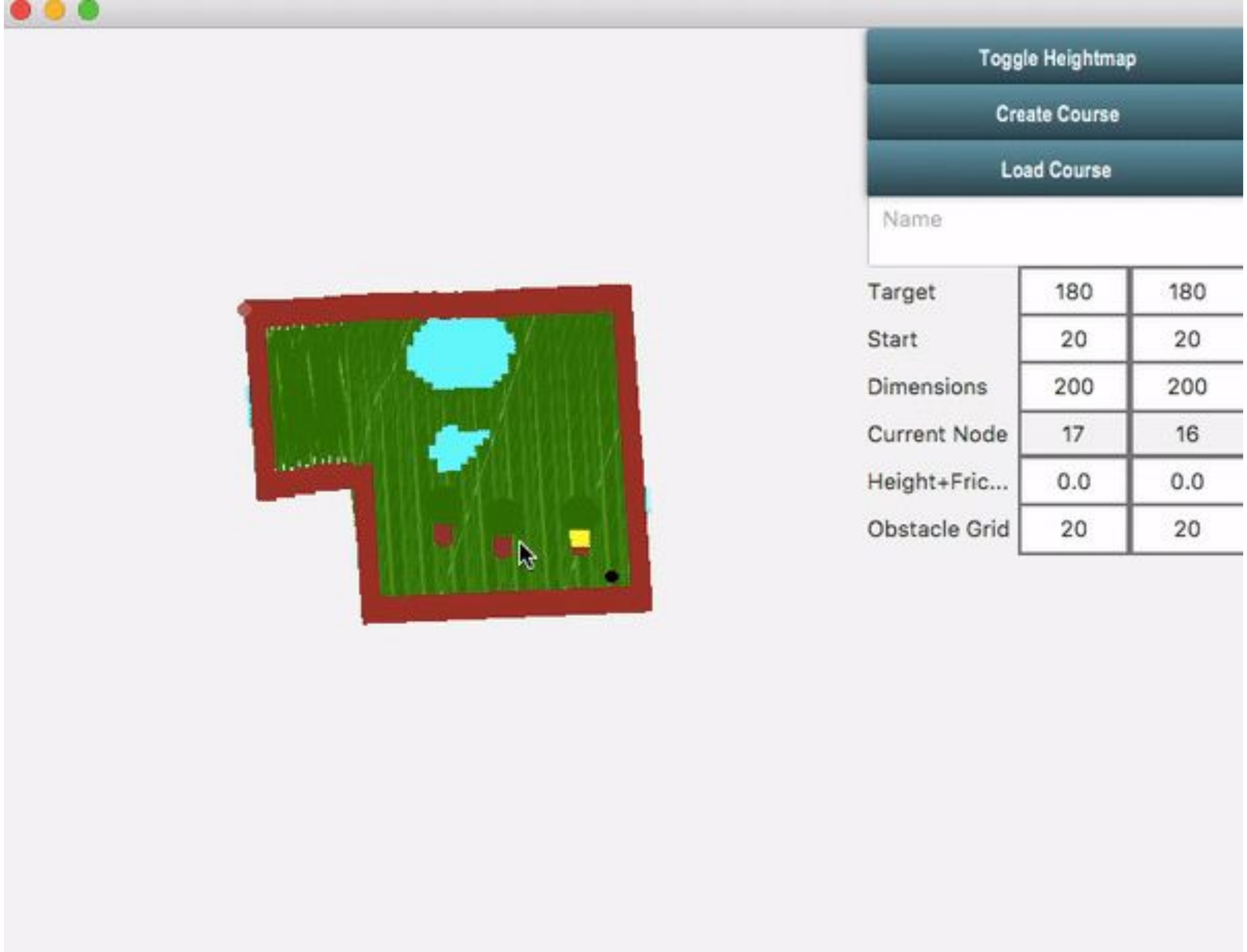
$$p_x(1,1) = \frac{p(0,3) - p(0,1)}{2}$$

Visualisation Techniques

- JavaFX(Triangle Mesh, FXML)
- Features:
  - course creator
  - heightmap
  - minimap
  - level saving, loading






Toggle Heightmap
Create Course
Load Course
Name
Target 180 180
Start 20 20
Dimensions 200 200
Current Node 17 16
Height+Fric... 0.0 0.0
Obstacle Grid 20 20

AI Algorithm

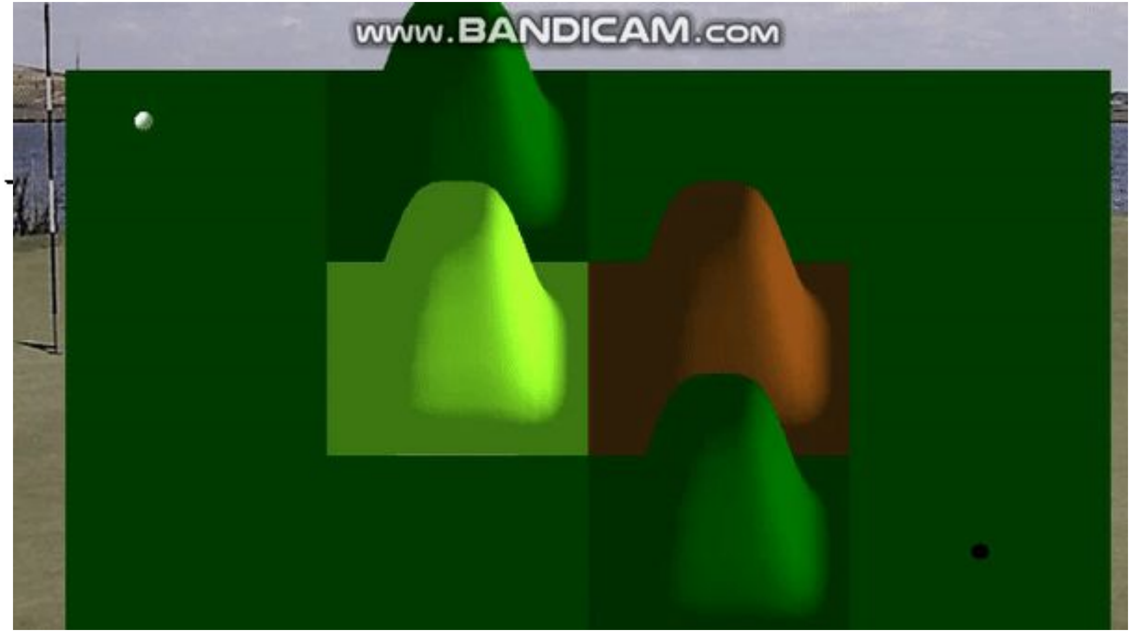
1. hilly surface - no obstacles
2. hilly surface - water - obstacles - maze
3. noise factor

+maximum hit force
+minimize operations(function evaluations) for search

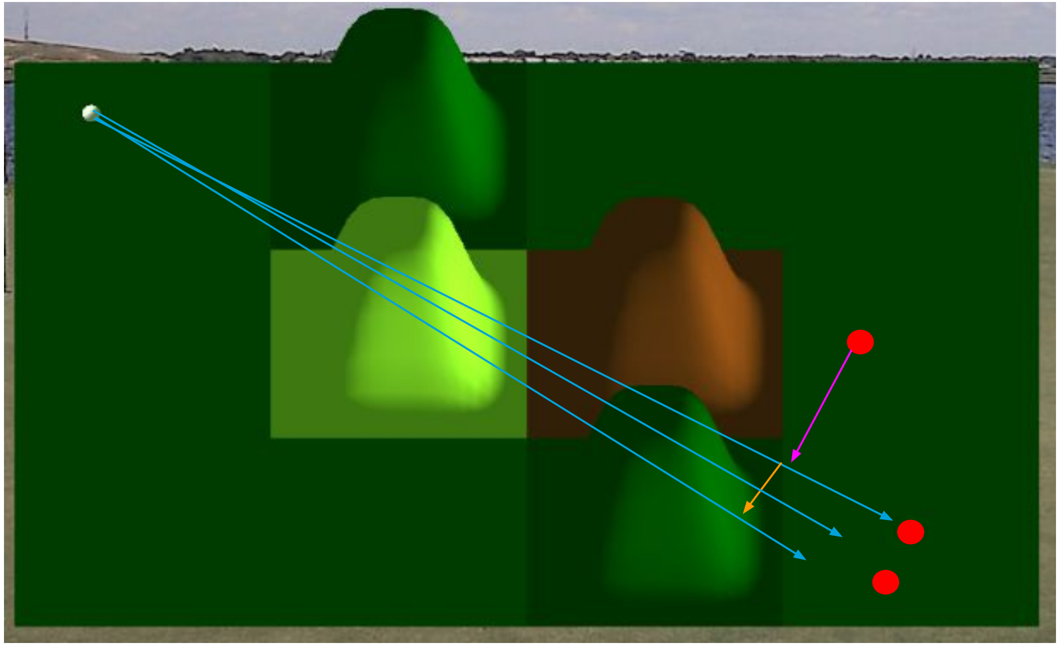
Hole-In-One without obstacles

Continuous optimization problem

$$f(x) : \mathbb{R}^2 \{|x| < V_{max}\}$$



Local Search/Geometric Heuristic



```
V → compute initial shot

while solution not found:
        S ← simulate(V)
        d ← deviation(S)
        if d < 0:
                V ← leftOf(V,dV)
        if d > 0:
                V ← rightOf(V,dV)
        if d = 0:
                V ← incForce(V,dV)
        if d = last d:
                 decrease stepsize dV
end
```

deviation $\leftarrow \vec{X}_{start-closest} \times \vec{X}_{start-hole}$

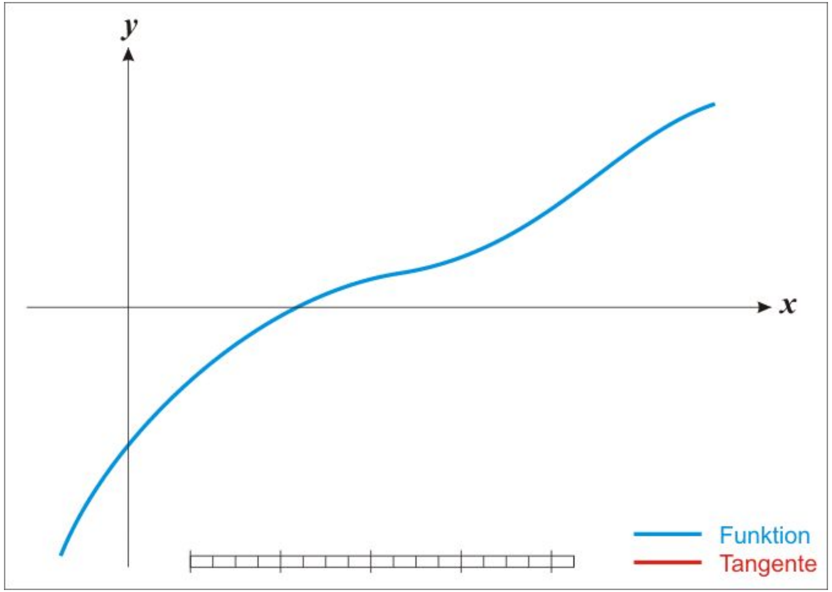
Newton's Method

- euc.dist. = f(x) = 0 ← find root input vector x
- iterate until root is found
- approximate partial derivatives by *finite differences(centred)*

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} f'''(\xi)$$

- small step size (0.0001)
- *can* fail:
  - stationary start
  - cycle
  - not continuous close to root

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



Particle Swarm Optimization

- population-based - global
- improve candidate solution
- exploitative behaviour

Each particle i has:

position $\vec{x}_{i,t}$ velocity $\vec{v}_{i,t}$ fitness $f\left(\vec{x}_{i,t}\right)$ personal best $\vec{p}_{i,t}$

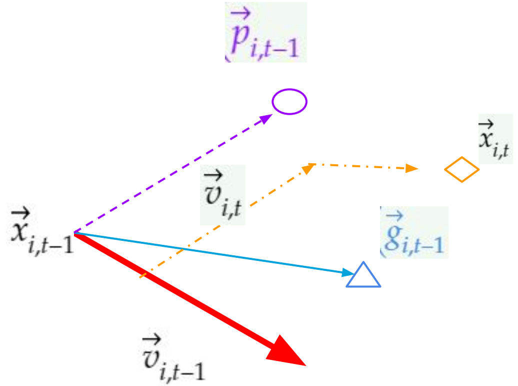
global best: $\vec{g}_{i,t}$

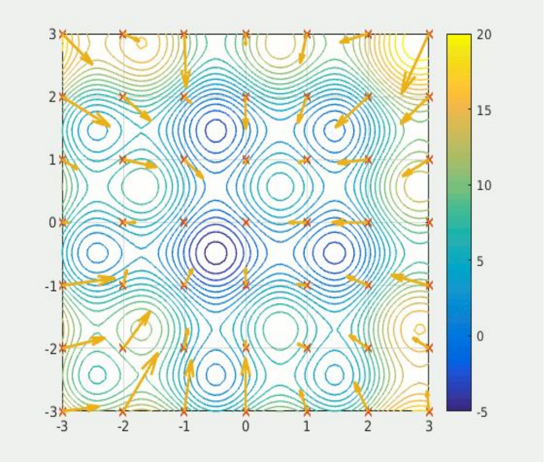
iteration step:

intertia term + *cognitive component* + *social component*

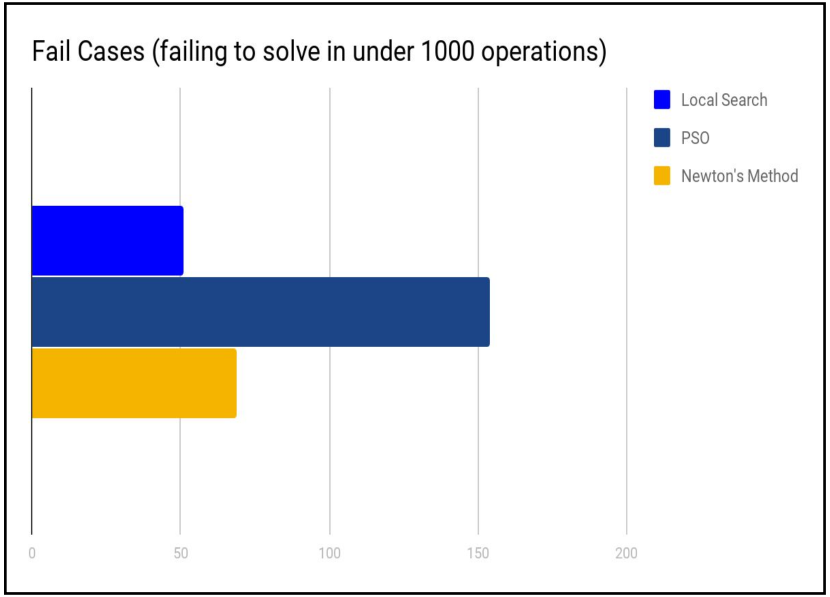
$$\vec{v}_{i,t} = \omega \cdot \vec{v}_{i,t-1} + c_1 \cdot \vec{r}_1 \times \left(\vec{p}_{i,t-1} - \vec{x}_{i,t-1}\right) + c_2 \cdot \vec{r}_2 \times \left(\vec{g}_{i,t-1} - \vec{x}_{i,t-1}\right); \vec{x}_{i,t} = \vec{x}_{i,t-1} + \vec{v}_{i,t};$$

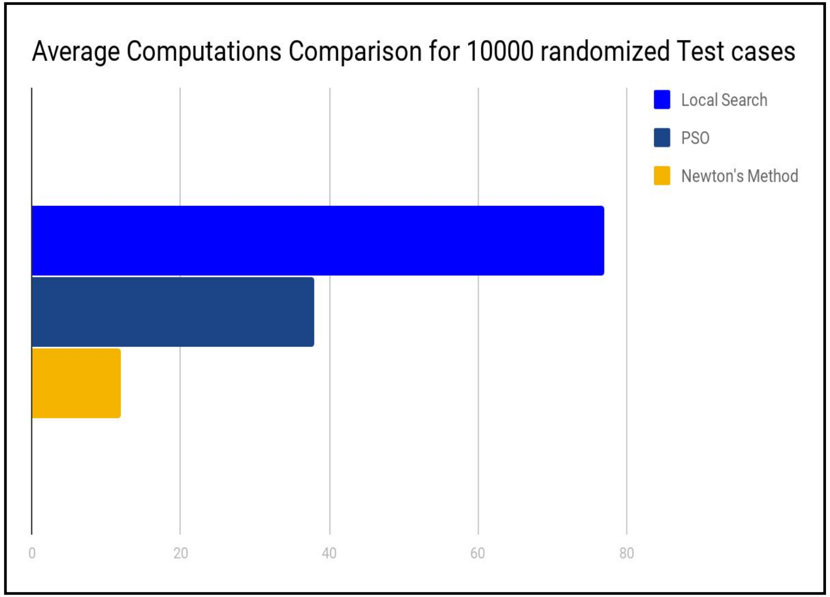
constant parameters: c_1, c_2; damped parameter: ω; random parameters $U\{0,1\}\vec{r}_1\vec{r}_2$;





Comparison - Testing - No Obstacles

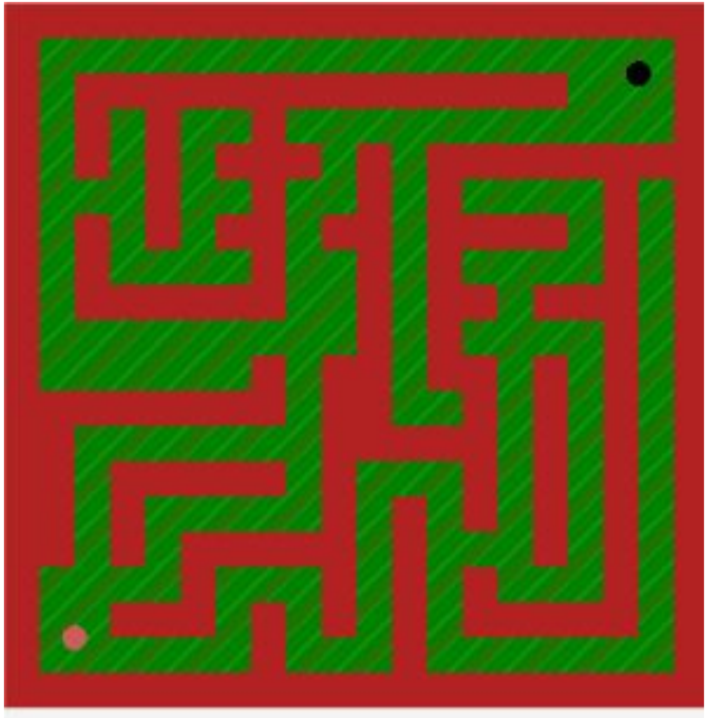




Hole-In-One with obstacles

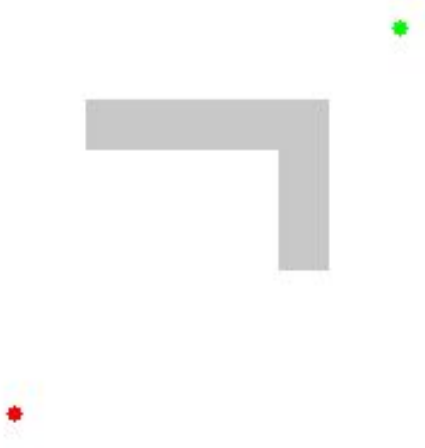
- Mazes, objects and water

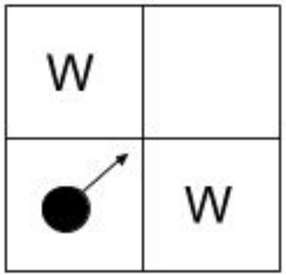
- Discontinuous function

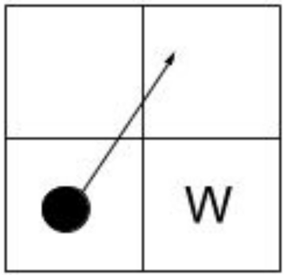


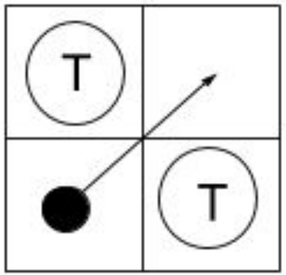
Pathfinding - A*

- Find shortest path through maze
- Obstacle grid
- Water
- g-value: amount of visited nodes
- h-value: minimal amount of nodes to reach goal
- f-value (g + h) used for Priority queue









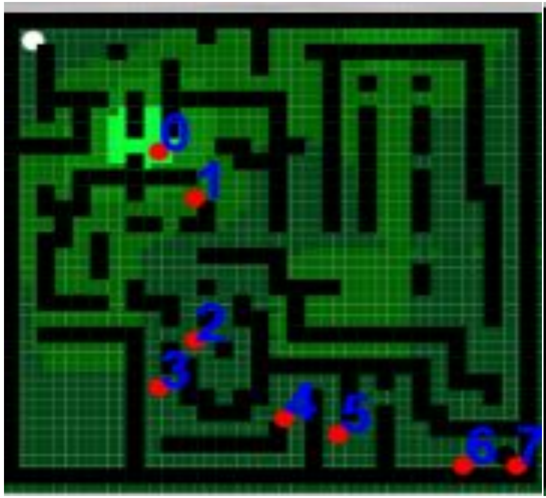
Competition

- Group splits into two teams
- Bots compete on several courses
- Results measured in number of iterations(function evaluations)

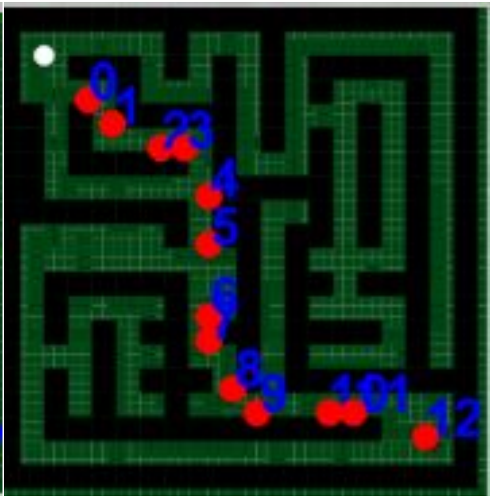
Algorithm A

Newton's Method + Pathfinding

```
path <- findPathA*
reducePath
initialGuess = 0
for each node in path:
        for small amount of iterations:
        initialGuess <- Newtons(node, initialGuess)
shot <- Newtons(target,initialGuess)
```



[critical points]



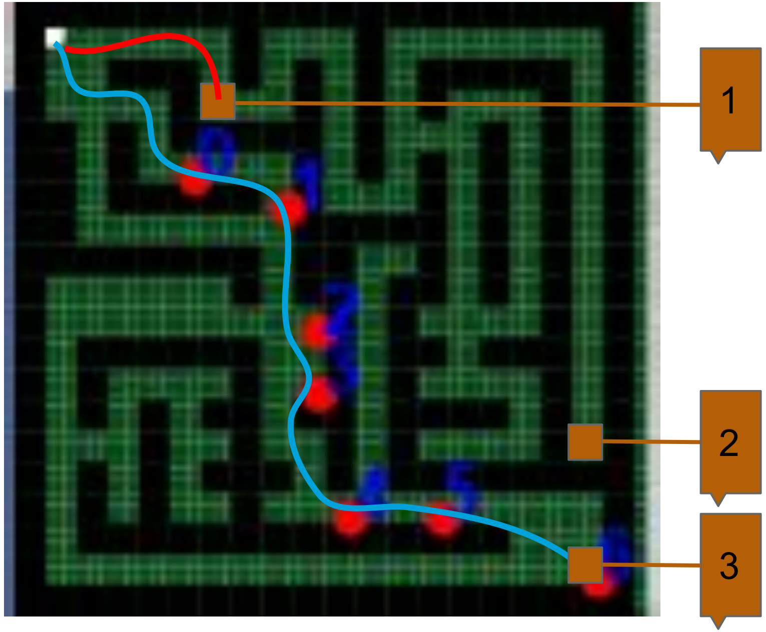
[50% of points]

local guided search

Algorithm B: PSO & Newton

- Difficulty in local search
- Euclidean in maze $\longrightarrow$ suboptimal?
- Path distance
- Below given threshold - zoom in
- Newton - Euclidean vs Path

Algorithm B: Path Distance



$$(size(p) - 1) - index(n_{b,p}) + \frac{d(pos(b), next(n_{b,p}))}{d_{max}}$$

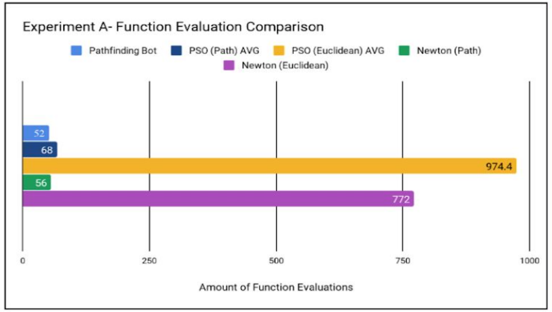
where $p = path$
$b =$ ball
$n_{b,p} =$ node of b in p
$d(x, y) =$ Euclidean distance between x and y
$d_{max} =$ maximum $d(x, y)$ possible in environment
$next(a) =$ the node after a in p
$h =$ hole

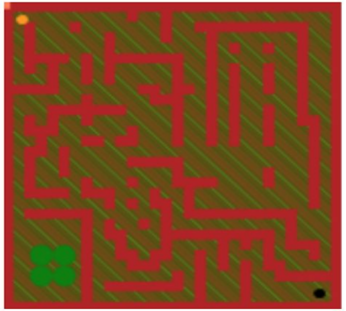
Noise

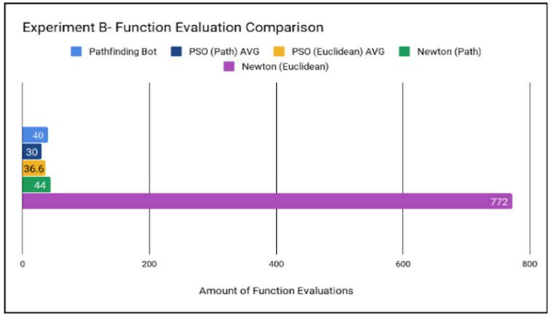
- Small position + velocity change
- More walls $\longrightarrow$ more unwanted collisions
- More water $\longrightarrow$ more failcases
- Player Bot
- “Safe Areas”

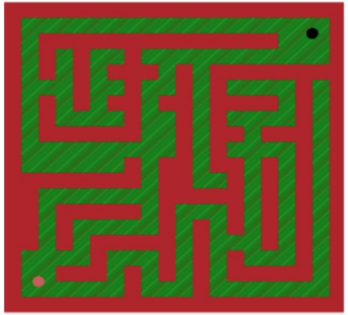
Experiment Overview

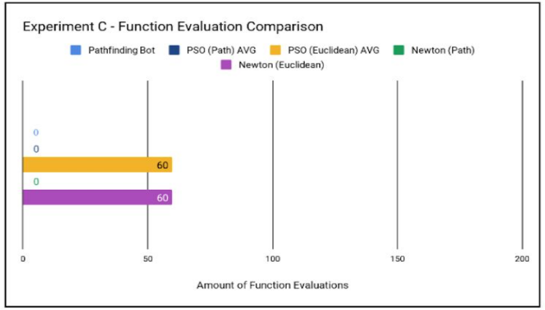
Pathfinding Bot | PSO (Path) AVG | PSO (Euclidean) AVG | Newton (Path) | Newton (Euclidean)

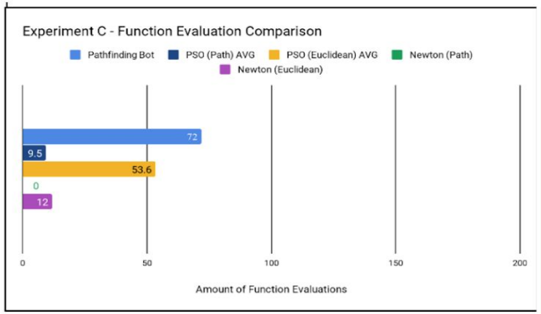


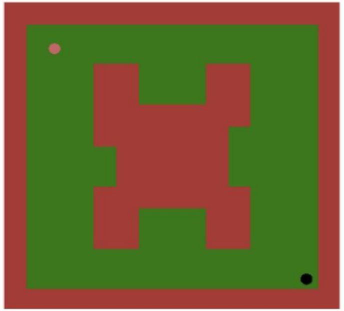


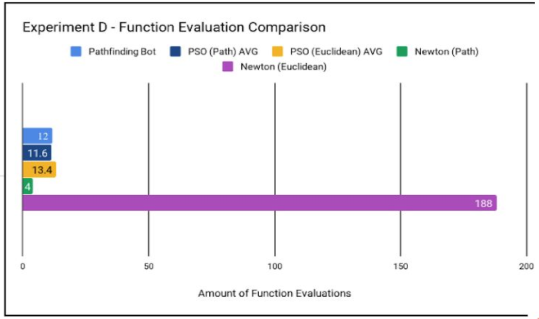


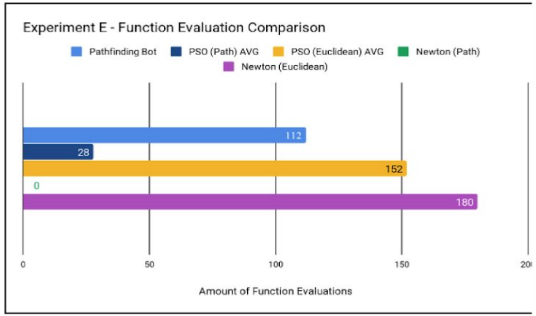


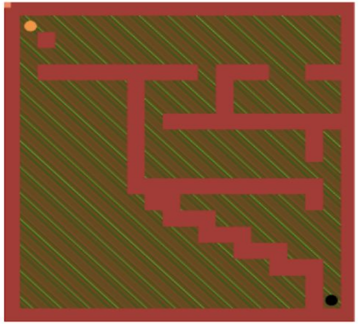




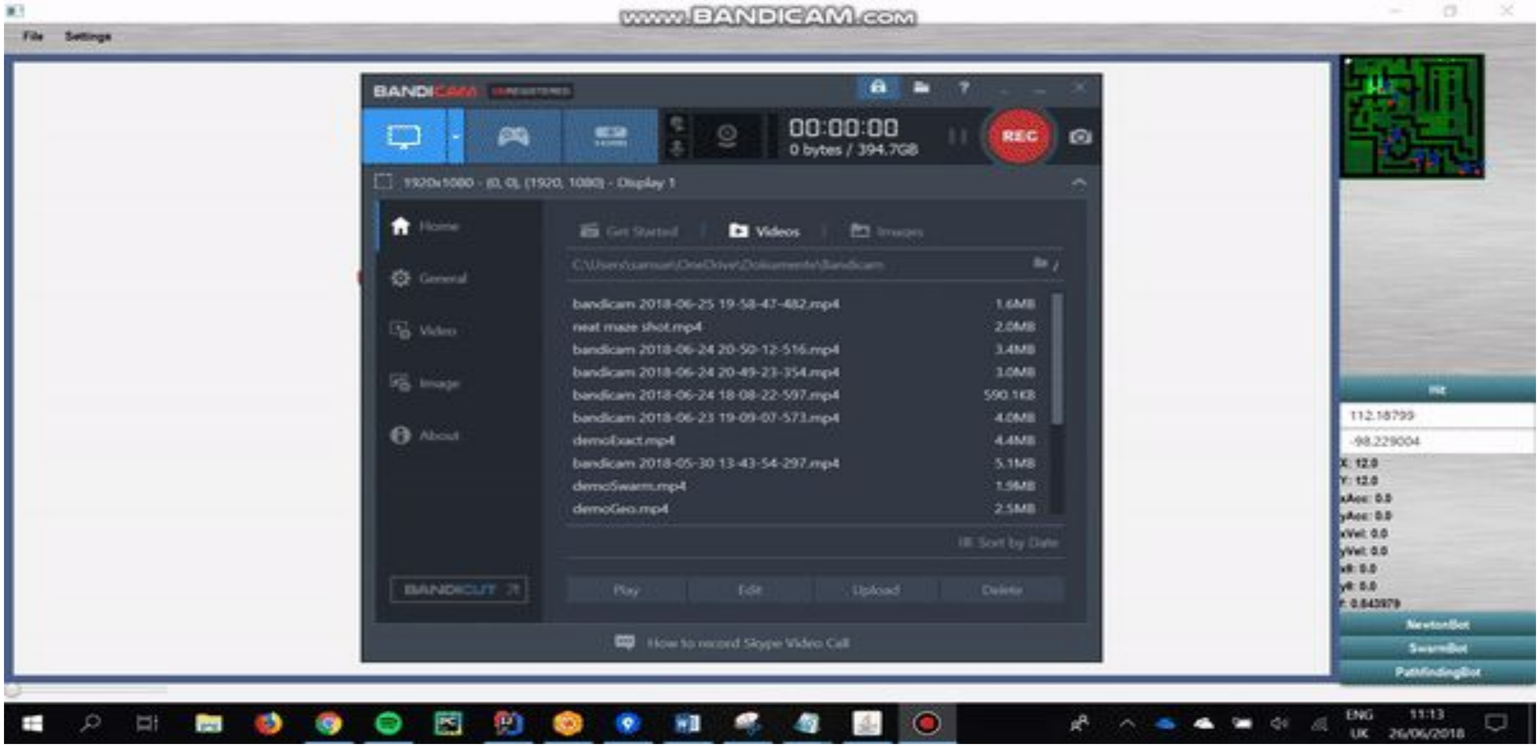




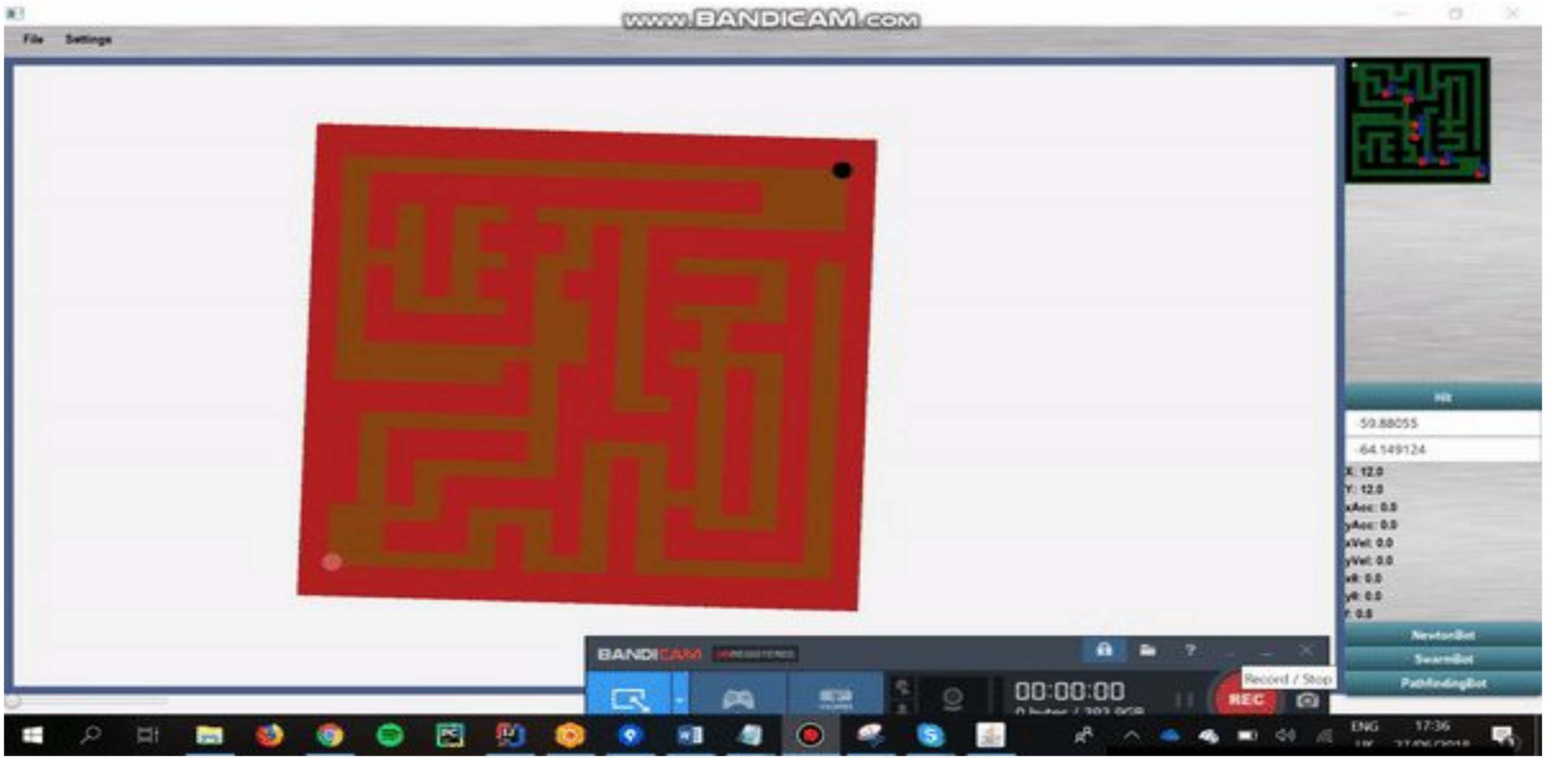




Experiment A - Algorithm A



Experiment B - Algorithm A



Conclusion

Research Question 1

In which course design does the performance of each algorithm stand out?

What are the course layouts for which the bot cannot find a solution?

Research Question 1

- simple mazes: the PSO outperforms the Pathfinding Bot

- complex mazes: both perform equally well

- water blocks the way $\rightarrow$ problems converging

Research Question 2

How do the algorithms handle a random noise factor? Up to which amount of noise do the algorithms still find a solution?

- the larger the mazes, the more difficult
- reason: more walls $\longrightarrow$ more collisions; deviation increases after each deflection
- 0.1% noise is manageable for small mazes

Research Question 3

How much does the pathfinding algorithm A actually improve the performance of the Bots in a maze environment?*

- complicated maze: <u>very</u> beneficial to use A*
- <u>BUT</u>: few obstacles: better to make no assumptions about path

Future Improvement

- Water detection
- More robust to non-continuity(water) alternative to Newton’ Method
- Physics → take ball's momentum into account