Crazy Putting Phase 3

RENÉ STEEMAN

AARON SCHAPIRA

IVAN POLIAKOV

JEAN JANSSEN

MATTHIJS KUSTERS

HAORAN LUAN

The challenge

1. Implement collision detection.
2. Improve the bot to handle complex terrains like mazes.
3. Having a random error in the initial position and velocity of the ball, including an analysis of the impact on the bot's performance.
4. Allow for balls that can both fly and bounce, as well as improvements to the bot so it can handle these new options.
5. Handling different (unknown) coefficients of friction and making sure the bot can handle them.

Table of contents

Game Engine

We have made our own openGL-based game engine with support for the following features:

Dynamic terrain generation

Real-time editing

Collision detection

Realistic lighting

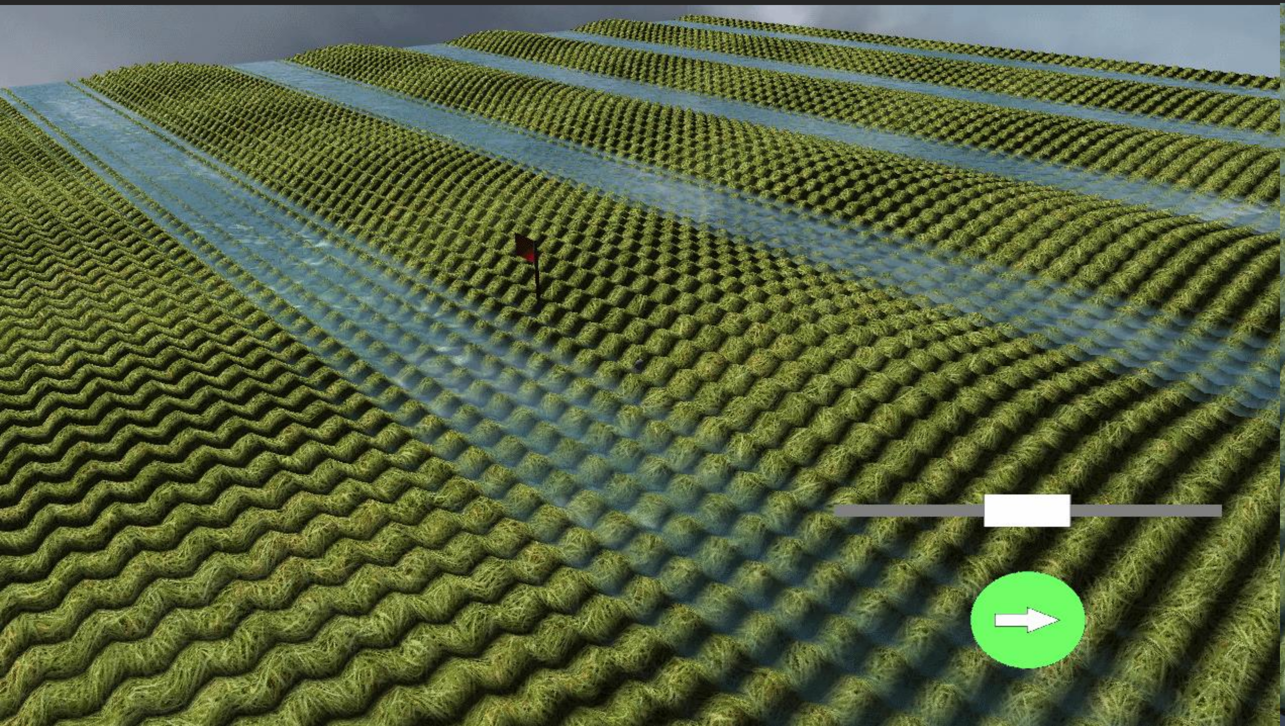
Water with special effects (du/dv maps, reflections, depth effect)

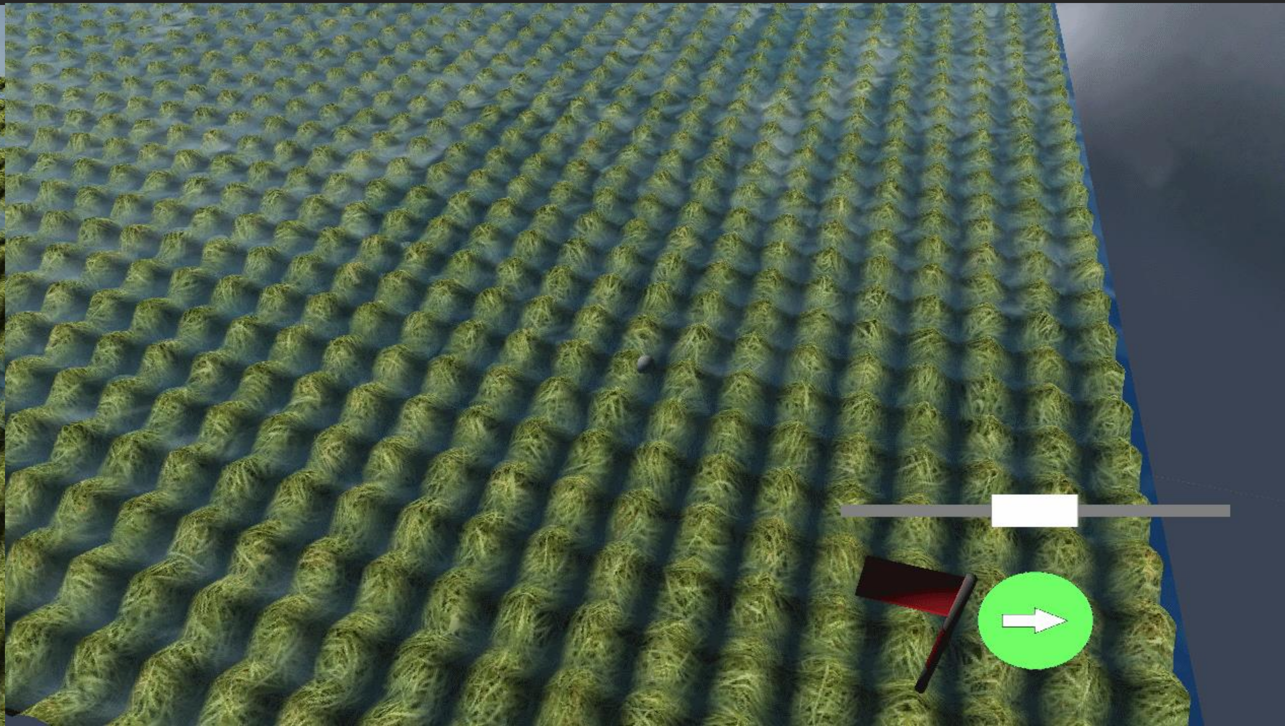
3D model support (.obj)

Third person camera

Terrain generation

Given a mathematical function with the variables x and y we can generate a terrain during runtime.



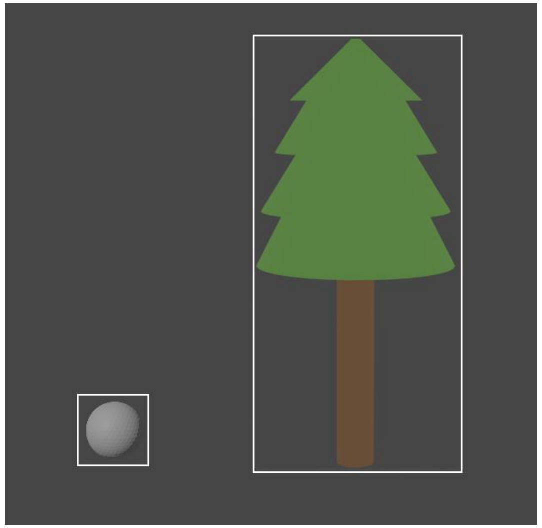


Collision detection

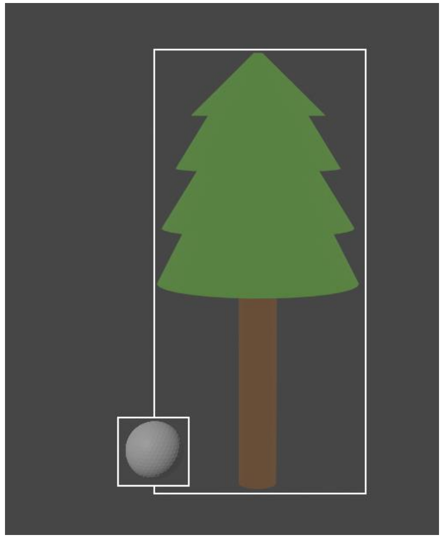
Stages:

1. Check if the smallest possible box around an object, that still contains every point making up the model, overlaps with that of the ball.

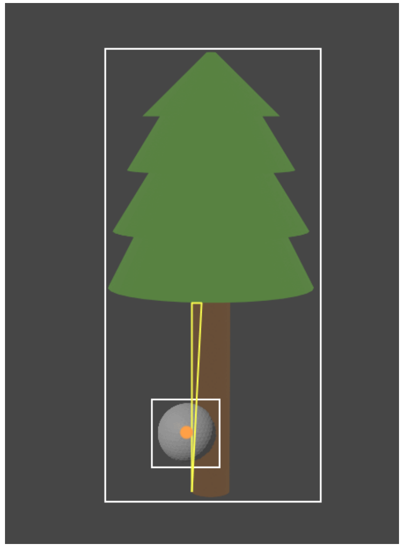
2. Check if the actual mesh is colliding with the ball



No collision

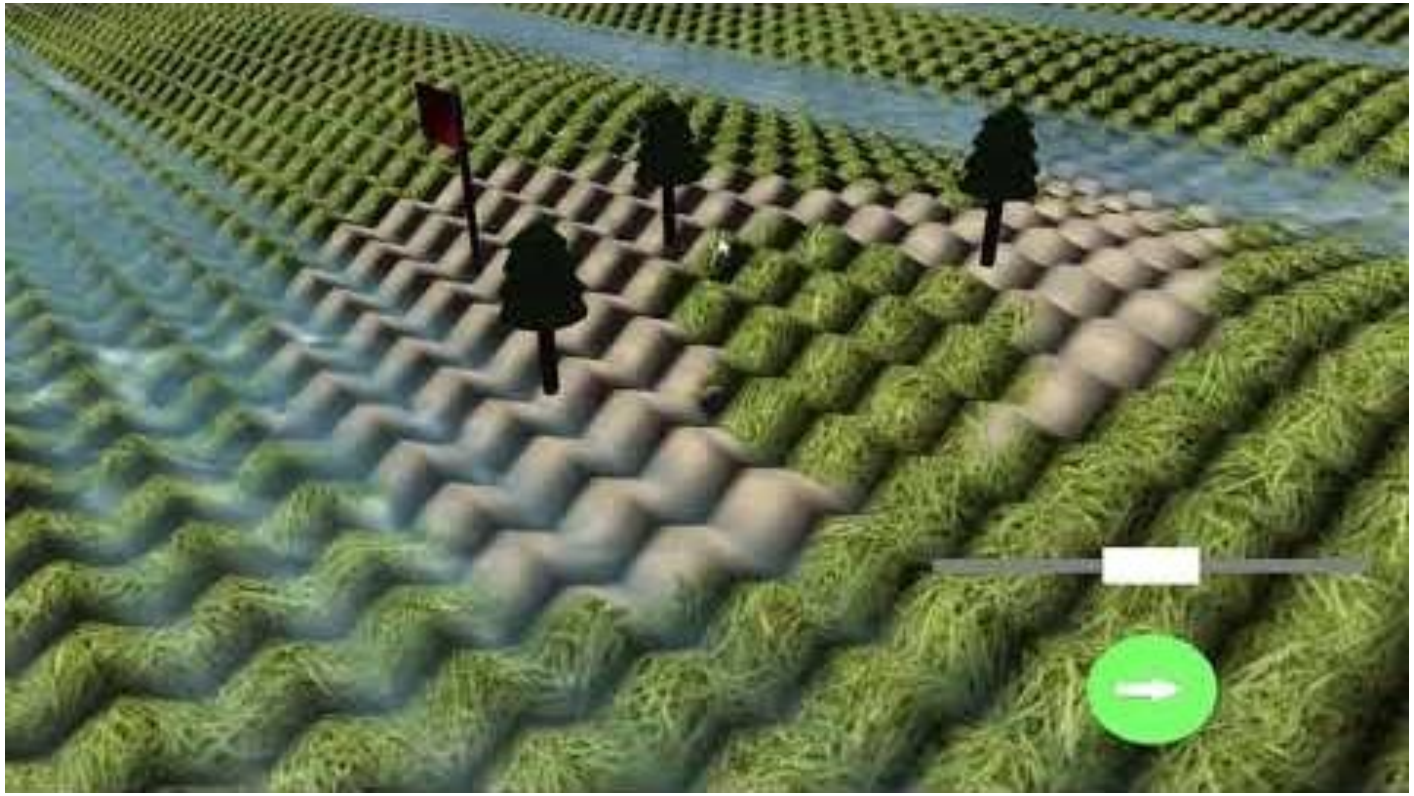


Possible collision



Collision

Live saving and loading



Our solvers

Last period we rewrote the entire physics engine to improve the interaction with the game. We also added 1 solver to improve the accuracy of our calculations.

- Classical 4th order Runge-Kutta

The formulas
Classical 4th-order Runge-Kutta

$$k_1 = v(t)$$

$$l_1 = a(p(t),\ v(t))$$

$$k_2 = v(t) + l_1 \frac{1}{2}\Delta t$$

$$l_2 = a\left(p(t) + k_1 \frac{1}{2}\Delta t,\ k_2\right)$$

$$k_3 = v(t) + l_2 \frac{1}{2}\Delta t$$

$$l_3 = a\left(p(t) + k_2 \frac{1}{2}\Delta t,\ k_3\right)$$

$$k_4 = p(t) + l_3 \Delta t$$

$$l_4 = a(p(t) + k_3 \Delta t,\ k_2)$$

$$p(t + \Delta t) = p(t) + \frac{1}{6}\Delta t(k_1 + 2k_2 + 2k_3 + k_4)$$

$$v(t + \Delta t) = v(t) + \frac{1}{6}\Delta t(l_1 + 2l_2 + 2l_3 + l_4)$$

Flying and bouncing balls

```
process(double deltaTime, (Vector position, Vector velocity))
    FOR time=0 TO time=deltaTime
        IF !isFlying(position)
        THEN velocity = redirectVelocity(position, velocity)

        calculate next position and velocity
        update position and velocity

    return (position, velocity)
```

Bouncing ball video

Flying acceleration

- In the horizontal directions:

$$F_{net} = -F_D \quad \rightarrow \quad ma = -\frac{1}{2}C_D \rho A\dot{x}^2$$

$$a = -\frac{C_D \rho A\dot{x}^2}{2m}$$

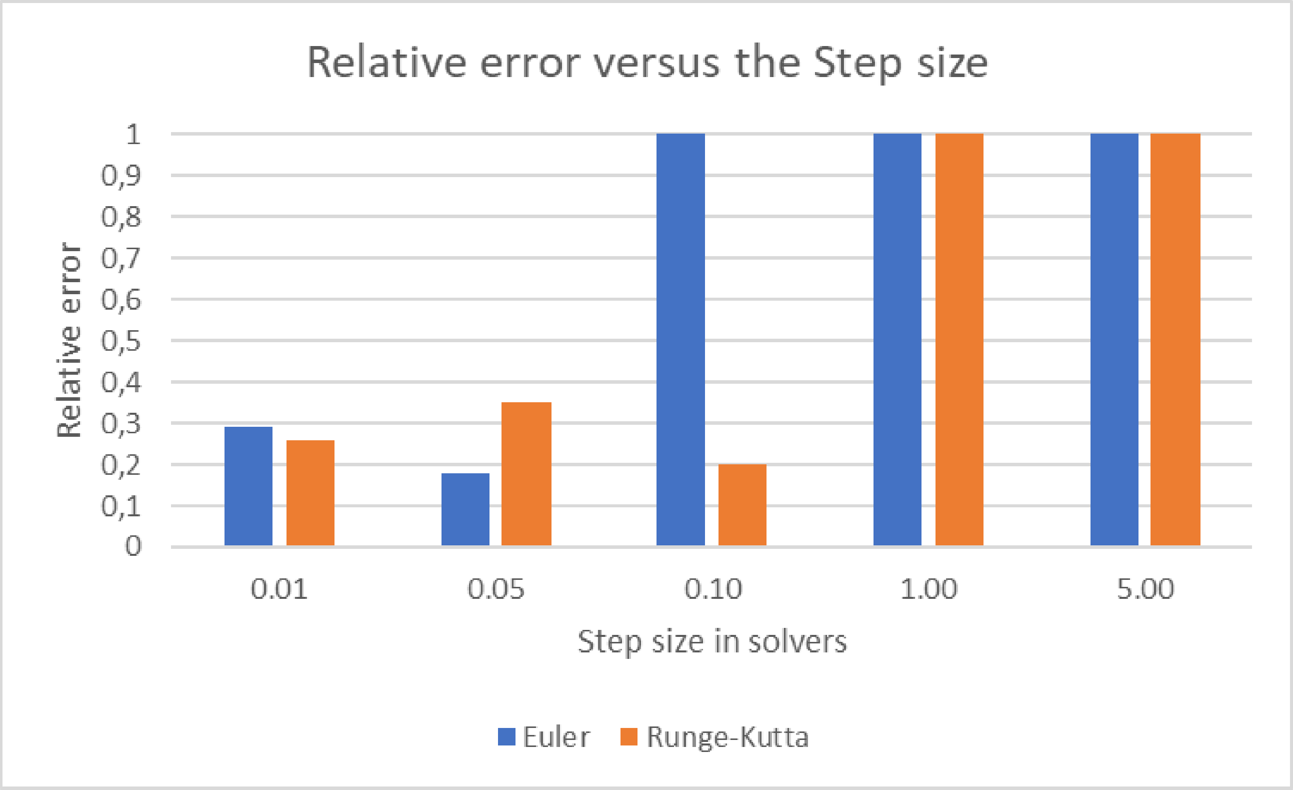
$$\rightarrow \quad a = -\frac{C_D \rho A\dot{y}^2}{2m}$$

- In the vertical direction:

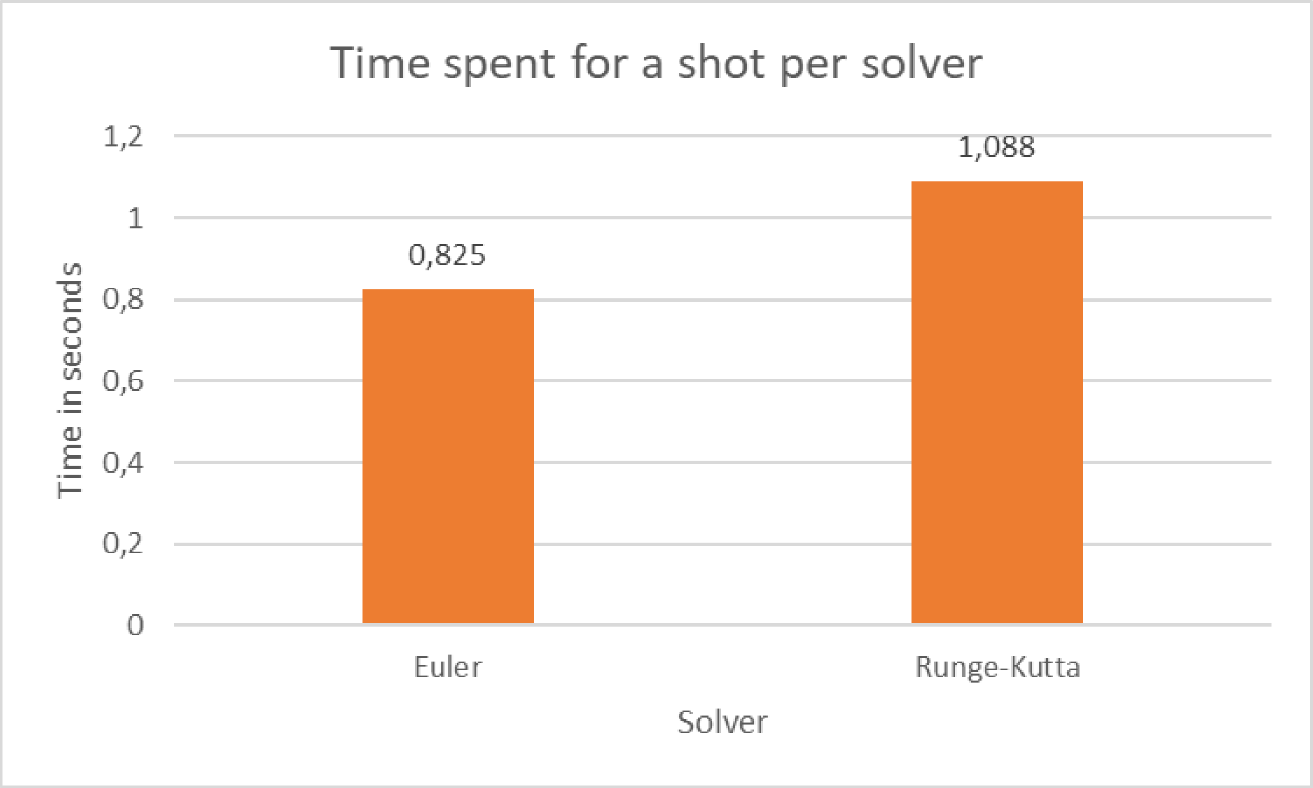
$$F_{net} = -F_g - F_D \quad \rightarrow \quad ma = -mg - \frac{1}{2}C_D \rho A\dot{z}^2$$

$$a = -g - \frac{C_D \rho A\dot{z}^2}{2m}$$

Precision experiment



Speed experiment



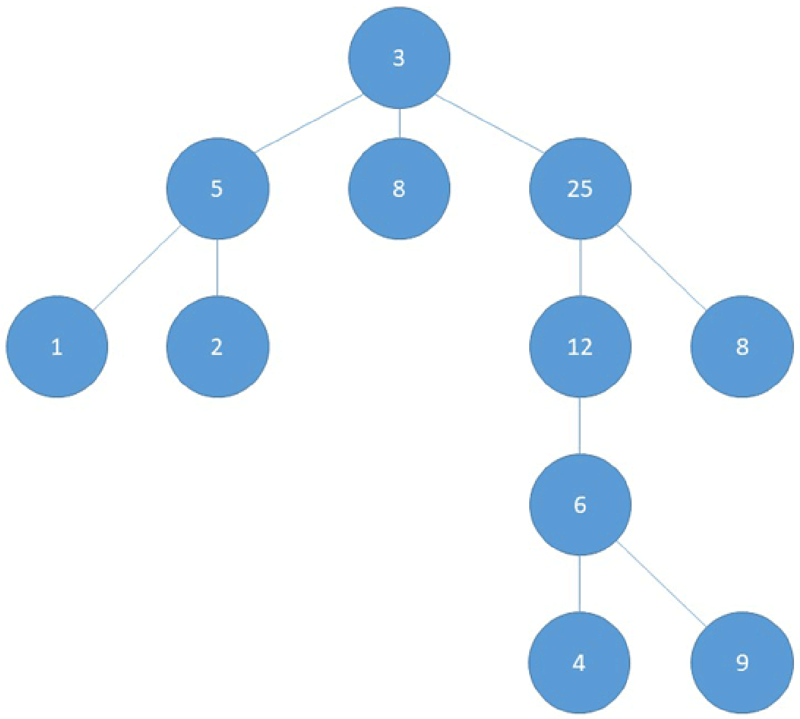
Bots

- Random error
- Different coefficients of frictions

Single Shot Bot

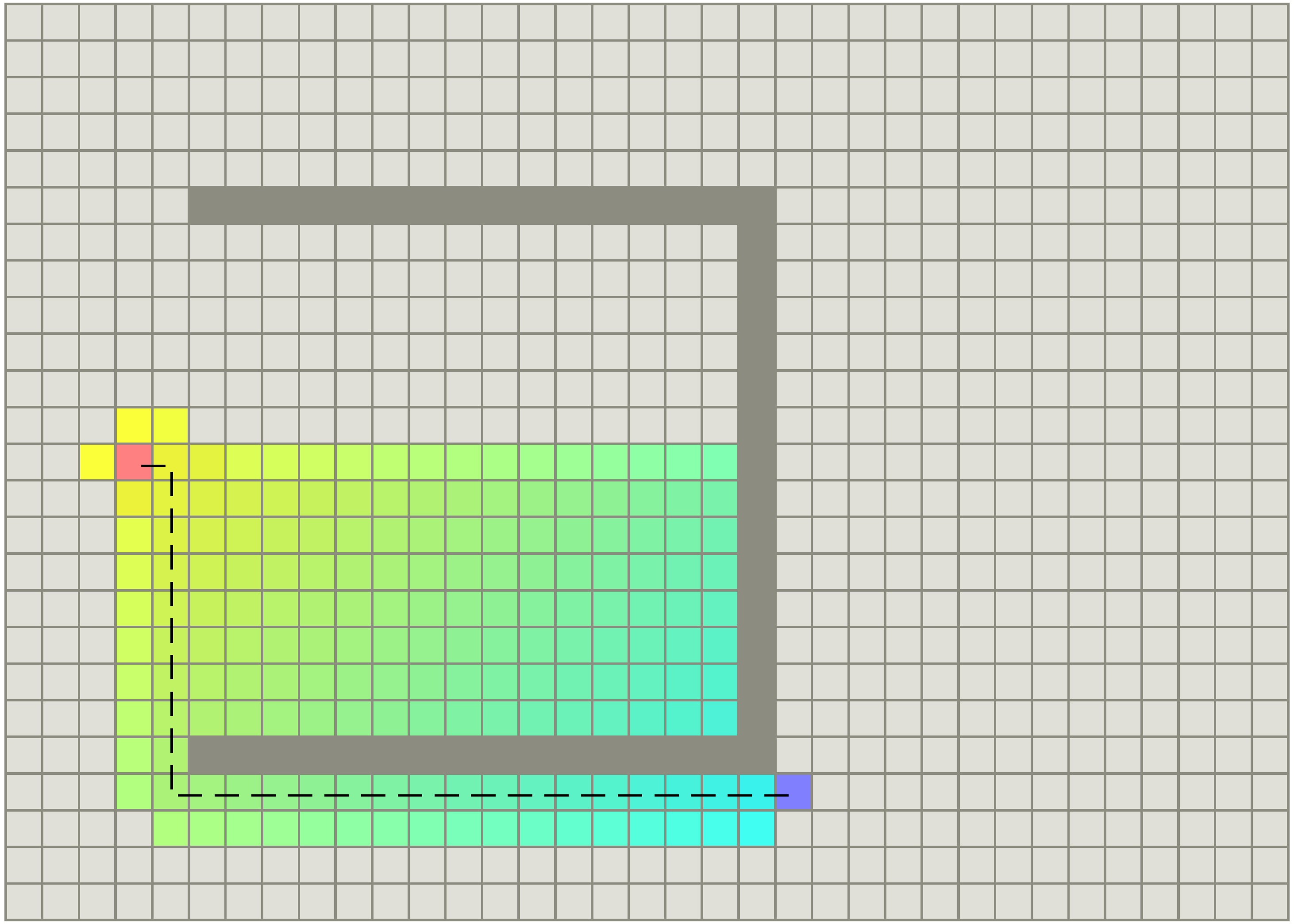
- Finds the location of the flag
- Computes shots in the area of the flag and stores the successful ones
- For each successful shot stored, it aims to try the perfect velocity
- If there is an obstacle between the ball and the flag, it will shoot the ball with a wider angle
- If no hole in one solution is possible, it won’t find a solution.
- O(maxVelocity / (degree step * velocity step))

BFS Graph Bot



Olivera Popović. (2020). Breadth-First Search [gif]. Stack Abuse https://stackabuse.com/graphs-in-java-breadth-first-search-bfs/

O(grid * the number of outcoming shots * shot complexity + grid)



From http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html by Stanford, date unknown

A* Bot

Experiments

Conclusion

- Graphics engine
- Collision detection
- Physics solvers
- Flying and bouncing balls
- AI

Questions

Sources

[1] Inc Epic games. Chaos Destruction, accessed 10-06-2020. https://docs.unrealengine.com/en-US/Engine/Chaos/ChaosDestruction/ index.html.

[2] The AlphaStar. AlphaStar: Mastering the RealTime Strategy Game StarCraft II, accessed 10-06-2020. https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii.

[3] Cass Everitt. Projective Texture Mapping [white paper], 2001. https://www.nvidia.com/en-us/drivers/Projective-Texture-Mapping/.

[4] Rene Truelsen.Real-time Shallow Water Simulation and EnvironmentMapping and Clouds. Universitetsparken 1, DK-2100 Copenhagen, Den-mark.

[5] Department of Data Science and Knowledge Engineer Maastricht University. Crazy Putting, 2019-2020.

[6] R.E.A. Bouwens, P.A.M. de Groot, W. Kranendonk, J.P. van Lune, C.M.Prop-van den Berg, J.A.M.H. van Riswick, and J.J. Westra. BiNaS.Groningen/Houten, The Netherlands, 6 edition, 2013. ISBN 978-90-01-81749-7.

[7] Anastas Ivanov and Juliana Javorova. THREE DIMENSIONAL GOLFBALL FLIGHT, 2017.

[8] Matthias Teschner. Simulation in Computer Graphics Particles. Computer Science Department University of Freiburg.https://cg.informatik.uni-freiburg.de/course_notes/sim_02_particles.pdf.

[9] Vadim Ivshin.Algorithm A* implementation.https://habr.com/en/post/331220/

[10] ITMO University.Algorithm A*.http://neerc.ifmo.ru/wiki/index.php

[11] PATH FINDING: A* OR DIJKSTRA'S?http://www.hindex.org/2014/p520.pdf

[12] Christian Holm. Simulation Methods in Physics 1. Institutefor Computational Physics University of Stuttgart, 2012-2013.https://www2.icp.uni-stuttgart.de/~icp/mediawiki/images/5/54/Skript_sim_methods_I.pdf.