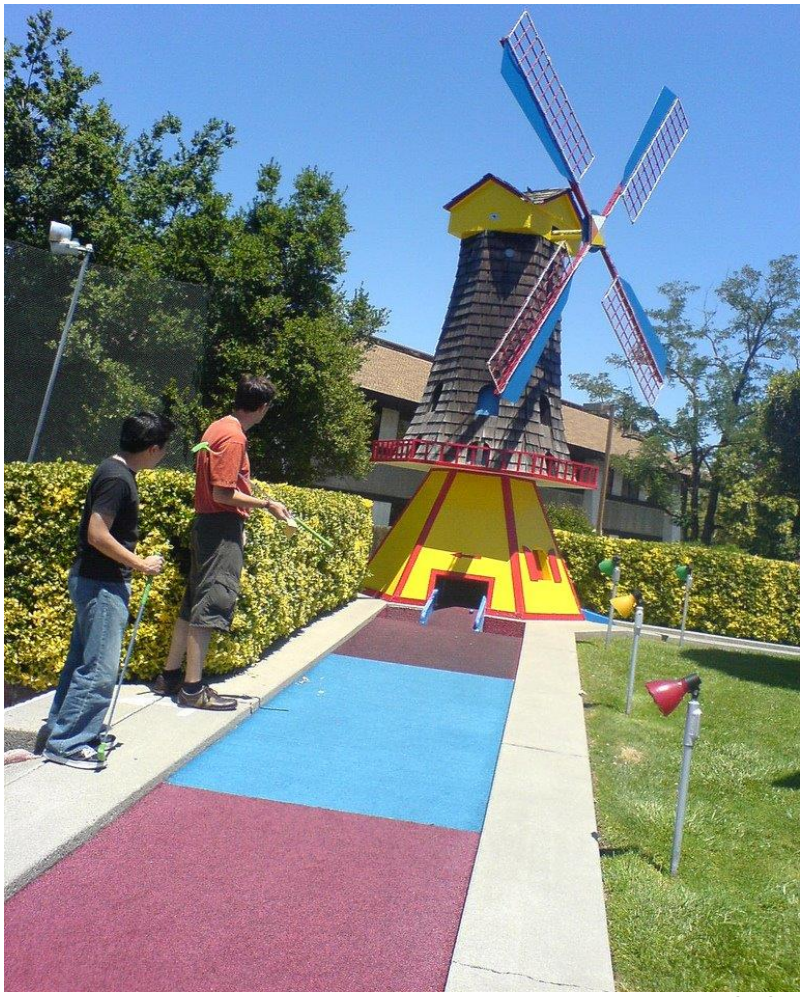
Project Manual Bachelor Year 1

Project 1-2

Crazy Putting!



Period 1.4, 1.5, and 1.6
Academic year 2019-2020
Data Science and Knowledge Engineering
Faculty of Science and Engineering
Maastricht University

Courses

Data Structures and Algorithms (KEN1420)
ICT and Knowledge Management (KEN1430)
Calculus (KEN1440)
Software Engineering (KEN1520)
Logic (KEN 1530)
Numerical Mathematics (KEN1540)

Project 1-2

The central topic of Project 1-2 is to create a simulator and artificial intelligence for putting on steeply sloping greens with obstacles.

1 Project description

Background

Putting is an important part of one of the world's most popular and exciting sports, namely golf [1]. In putting, players aim to strike a ball across a grassy green field to have it fall into a hole. This is made difficult by the fact that the green is not necessarily flat, requiring the player to accurately judge how the slope will affect the motion of the ball. In some cases, the green may also contain obstacles such as sand pits, trees or even water.

Objective

You have been hired by a game company to create a Crazy Putting simulation program, which should capture all the excitement of the real-life putting experience from the comfort of your student common room. The program should realistically simulate the physics of real-life putting, and allow for players to play against each other or an artificial intelligence player (bot). The program should also allow users to create their own putting greens, complete with gentle rolling hills, steep mountain valleys, and even sand bunkers or deep lakes of water (the final resting place of many a mishit golf ball).

Physics

Each time it is hit, the golf ball is set in motion in a given direction, with the speed determined by how hard it is hit, up to some given maximum speed v_{max}.

To simplify the physics of the ball we will assume that the ball is always sliding on top of the terrain. You do not have to simulate rolling motion of a ball, or consider flying motion.

The motion of the ball across is governed by two forces:

1. The force of gravity, as determined by the sloping landscape.
2. The force of friction, as determined by the terrain.

The motion is described by a system of differential equations, given after the main project description.

Terrain

The terrain is described by a function h giving the height $z = h(x, y)$ at every point, and coefficient of friction μ. Further, the terrain may have bodies of water, which are found where the height z is negative. You should allow for additional obstacles, such as sand pits (which have a higher μ), or trees (which the ball bounces from with a *coefficient of restitution* λ).

The Game

The ball has to make its way across the green from its starting point to the target, while avoiding any pools of water. For the purposes of this game, instead of a hole, the player aims to strike the ball so that it *stops* within a given radius r of a marked target.

The player aims to guide the ball into the target in as few shots as possible.

If the ball rolls into the water, the player incurs a one-stroke penalty, and chooses either

i. Replays the shot from the initial location, or
ii. Play at any point such that the place where the ball fell into the water is directly between that point and the hole.

See [1, Rule 26] for the full regulations. As a first implementation, you may choose to only implement i. or only allow a restart where the ball fell into the water.

Warning: When solving the equations of motion, you will probably use a time-stepping method, computing positions at finitely many time instances. However, you should take precautions to ensure that the ball does not fall into the water at times *between* the computed time instances.

The Software

Make sure you separate the physics engine, the graphical representation, the AI and the data structures that contain your game elements. You are not allowed to use existing physics engines or mathematical solvers, but must implement these yourself. However, you are allowed to use an existing graphics library. We recommend using LibGDX as the graphics and game engine library, but you are allowed to use other software.

The Project

The project consists of several design and developing tasks, organised in three main phases. The main phases are defined:

I. Game Engine

In the first phase, you set up the game engine, including a course designer

Tasks:

1. ~~Implement a physics engine for the crazy putting game. The engine must be able to handle sloping courses, and friction following the laws of physics given in the Appendix. You should implement the Euler differential equations solver described in the Appendix.~~
2. ~~Implement a course input-output module, allowing for input and output of your courses from/to a file. The input-output module should allow input and output of courses, including the example course given later.~~
3. Implement a simulator with at least two game modes: (1) In mode 1 a human player should be enabled to play by setting the initial velocity (speed and direction) of each shot. (2) In mode 2 the initial velocity (speed and direction) of each shot should be read from a file.
4. ~~Implement a course designer, allowing for the height profile and friction coefficient of the course to be specified (by an explicit function e.g. $z = \sin(x) + y^2$), the start and goal positions of the ball, the radius of the target, and the maximum velocity.~~

II. Artificial Intelligence

In the second phase, you implement an artificial intelligence player capable of analysing the course and computing how to strike the ball.

Tasks:

1. Improve your physics engine by implementing solvers based on (i) the second-order Verlet solver, (ii) the classical 4th-order Runge-Kutta solver, and (optionally) any other solver you choose.
2. Add an artificial intelligence player capable of playing on the terrain you create. The bot may have full knowledge of the course, including the geometry and slope

of the terrain, and the physics of the ball motion. The bot should consistently manage to score a hole-in-one, where possible, but does not need (yet) to handle complex, maze-like courses.

III. Advanced Topics

In the third phase, you focus on improvements to the AI for puzzle-like aspects of the game and look at some extensions for the game.

Tasks:

1. Allow obstacles and bodies of water in your course design. Implement a collision-detector for determining whether the ball has fallen into a body of water or hit an obstacle, and handle these appropriately.
2. Extend your AI bot player to handle complex maze-like courses with obstacles. Design and test your bot on courses which allow you to identify the limits of your bots capabilities.

In addition, choose at least one of the following advanced problems:

3. Cubic splines are widely-used to model curved surfaces. Improve your course designer by allowing the use of splines to describe the height of the course.

4. Humans are not perfect, and neither should your AI bot be. Further, the world is uncertain, and players have to adjust to different conditions. Choose one (or more) of the following:
   a) Introduce a small random error in the initial position and velocity of the ball and investigate how this affects performance of your bot. Vary the magnitude of this error, and investigate how your bot can find strategies to deal with it.
   b) On a windy day, the ball may be buffeted around as it rolls. Extend your physics engine to produce a randomly varying force due to the wind, and create a bot to handle this.
   c) Different greens have different coefficients of friction. Extend your bot to handle an unknown coefficient of friction (which you can try to estimate).

   Can you design a course for which it is better for the bot to play with a different strategy to the noise-free case?

5. Make a more realistic and powerful physics engine. Choose one (or more) of the following extensions:
   a) The physics of rolling is actually more complicated than the simple model described here. Research the physics of rolling, and implement it in your physics engine.
   b) If a ball is hit hard on a sloping course, it may fly into the air. Extend your simulator to handle flying and bouncing balls. You could even design some “crazy golf” courses.
   c) Extend your simulator to handle the complete game of golf, including hitting the ball into the air from the fairway.

6. Organize a competition! Split your group into two teams, each of which should develop a bot and design two or three courses. Play over both sets of courses, and see whose bot wins! Of course, you may design courses to highlight your bot's strengths, but you may absolutely not pre-program the solution for the courses you design!

   Even better, play against another group! To do this, you will need ta common API for the courses and your bot. (We are not planning a public competition, but could try to schedule one if there is sufficient interest.)

The code

You must write your code in Java. You should pay attention to software design and engineering aspects, including data structures, class hierarchies, and code validation. You should also pay attention to proper project management and scheduling, including the creation of timelines, division of labour, and sharing, versioning and backing-up of code.

Write unit tests to test your code. You can use JUnit in Java. Report code coverage for your final software product.

We encourage you to write a build script using Maven or Grable. This will allow us to build and test your software automatically without installing any integrated development environments.

Use a Git repository to track of the code. We recommend using an online Git hosting facility, such as GitHub, Bitbucket, GitLab or SourceForge. You should set up the repository as soon as possible, and use more advanced features once they have been covered in the Software Engineering course. Use branches for major changes in the code, and use the hosting facilities to post issues and assign feature requests. We will take a look at your repository to evaluate the contributions of each team member.

The presentations

During presentations <u>all</u> team members have to talk. Make proper use of presentation material. Create video material as backup in case that live demonstrations do not work as intended. Properly prepare live demonstrations so that you do not lose unnecessary time. e.g., read course descriptions from a file to avoid setting up new courses during the presentation through a complex user interface.

Presentations should properly report on the progress of your work, provide proof that the items of the project description have been successfully implemented, honestly report about items that are not yet working, and end with a proper planning of the next phases.

The report

Your report should be a scientific document describing your work on the project. It should provide supporting background information, including references to published work. (Unless no other source is available, references should be published articles or books, and

not online blogs, wikis etc.) It should clearly describe what you accomplished yourselves. Direct use of existing text or figures must be properly attributed. (Where possible, draw figures yourselves using the facilities provided by your document preparation system.) Descriptions of the methods and algorithms must correspond to what is actually implemented in your code. Lack of proper attribution of work that is not yours, and discrepancies between what you claim to do in your report, and what you actually do in your code, will be heavily penalised.

In the report, you should write-out the complete equations of motion you use to describe the physics of the ball. You should carefully describe the algorithms you use for simulating the motion and testing whether the ball falls into the water or hits an obstacle. In all cases, you should use appropriate mathematical notation. You should also describe the algorithms used for the artificial intelligence, using mathematical equations and/or pseudocode if/when appropriate. Analyse the performance of your AI bots carefully, using experiments to showcase where your AI performs well but also to demonstrate where your AI fails. Provide explanations for the performance of your AI.

You must provide an appendix briefly listing the contribution of each team member to the project.

All team members have to contribute to writing the final report.

See the "Guidelines Report" for more details.

Grading

If you properly and competently implement, test, and report on the items of the project, you can expect a grade around 8.0. Higher grades can be achieved by particularly creative solutions (especially to the advanced tasks in Phase 3), with an exceptionally well-written report or clear presentation, and/or with especially insightful answers to questions.

See the evaluation assessment forms uploaded to student portal for a better understanding of the criteria for grading.

References

1. United States Golf Association (USGA) Rules and Decisions http://www.usga.org/content/usga/home-page/rules/rules-and-decisions.html
2. J. Tierney "Better Science Through Mini-Golf?", *New York Times*, 2009. http://tierneylab.blogs.nytimes.com/2009/06/15/better-science-through-mini-golf/
3. R.C. Hibbeler, *Engineering Mechanics: Statics & Dynamics*, Pearson, 2007.
4. J. Hierrezuelo, B. R. Catolicos, and C. Carnero, "Sliding and rolling: The physics of a rolling ball", *Physics Educ.* 30, 177–182, 1995.
5. A.R. Penner, "The physics of putting", *Canadian. J. Physics* 80, 2002.
6. R.A. Adams, Calculus: A Complete Course, Addison-Wesley
7. J.D. Faires and R. Burden, *Numerical Methods*, Brookes-Cole, 2013.

Appendices

Differential Equations

A *differential equation* is a way of describing the laws governing how a system changes in time [1]. A first-order equation specifies how the state p of a system at changes in time as a function of p itself, $\dot{p}(t) = f\big(p(t)\big)$, where $\dot{p}$ denotes dp/dt. Most differential equations occurring in physics are second order, which means they involve second derivatives $\ddot{p} = d^2\,p/dt^2$.

For example, the differential equation $\ddot{x} = 2x - \dot{x}$ has a solution $x(t) = 3e^{-2t}$ since

$$\dot{x}(t) = \frac{d}{dt}x(t) = -6e^{-2t}; \quad \ddot{x}(t) = \frac{d^2}{dt^2}x(t) = \frac{d}{dt}\dot{x}(t) = \frac{d}{dt}(-6e^{-2t}) = 12e^{-2t}$$

and

$$2x(t) - \dot{x}(t) = 2 \times (3e^{-2t}) - (-6e^{-2t}) = 12e^{-2t} = \ddot{x}(t).$$

Physics

We describe the position of a ball by its coordinates $p = \big(p_x, p_y\big) = (x, y)$, with velocity $v = \dot{p} = \big(v_x, v_y\big) = (\dot{x}, \dot{y})$ and acceleration $a = \dot{v} = \ddot{p} = \big(a_x, a_y\big) = (\ddot{x}, \ddot{y})$.

The physics of a ball moving on a hilly surface, of height $z = h(x, y)$ is defined as follows:

We require the partial derivatives

$$\frac{\partial z}{\partial x} = h_{,x}(x, y) \coloneqq \lim_{\delta x \to 0} \frac{h(x+\delta x, y) - h(x, y)}{\delta x}; \quad \frac{\partial z}{\partial y} = h_{,y}(x, y) \coloneqq \lim_{\delta y \to 0} \frac{h(x, y+\delta y) - h(x, y)}{\delta y}.$$

The gravitational force due to the slope is given by

$$G = \Big(-mgh_{,x}(x, y), -mgh_{,y}(x, y)\Big),$$

where m is the mass of the ball, and $g = 9.81\text{ms}^{-2}$ the acceleration due to gravity.

The force due to friction of a moving object is:

$$H = -\mu mg\, v/\|v\|$$

where $\|v\| = \sqrt{v_x^2 + v_y^2}$ and μ is the coefficient of friction.

The equations of motion are

$$\ddot{p} = a = F/m$$

where $F = F(p, \dot{p}) = F(p, v) = G + H$ is the total force applied.

These forces give rise to the complete differential equation describing the motion:

$$\ddot{x} = -gh_{,x}(x, y) - \mu g\, \dot{x}/\sqrt{\dot{x}^2 + \dot{y}^2}; \quad \ddot{y} = -gh_{,y}(x, y) - \mu g\, \dot{y}/\sqrt{\dot{x}^2 + \dot{y}^2}.$$

Euler's Method

In the *Numerical Mathematics* course, you will learn techniques for solving the differential equations of motion. For Phase 1, it suffices to use Euler's method:

You will need to solve the differential equation $\ddot{p} = a = F(p, \dot{p})/m$. The standard numerical approach is to reduce to a system of first-order equations by introducing the velocity $v = \dot{p}$ as a new variable, to obtain the system

$$\dot{p} = v; \quad \dot{v} = a = F(p, v)/m \, .$$

Then the updated position and velocity after a single time step of length h is

$$p(t + h) \approx p(t) + hv(t); \quad v(t + h) \approx v(t) + hF\left(p(t), v(t)\right)/m.$$

(Note that the 'h' used here for the step size is different to the usage as the height function; you may prefer to use the notation δt instead of h.)

In components

$$p_x(t + h) \approx p_x(t) + hv_x(t); \;\; v_x(t + h) = v_x(t) + hF_x\left(p_x, p_y, v_x, v_y\right)/m \, ;$$
$$p_y(t + h) \approx p_y(t) + hv_y(t); \;\; v_y(t + h) = v_y(t) + hF_y\left(p_x, p_y, v_x, v_y\right)/m.$$

Example Course

```
g = 9.81;   // Gravitational acceleration [m/s^2]
m = 45.93;  // Mass of ball [g]
mu = 0.131; // Coefficient of friction (rolling ball)
            // Typical 0.065<=mu<=0.196
vmax = 3;   // Maximum initial ball speed [m/s]
tol = 0.02; // Distance from hole for a successful putt [m]

start = (0.0, 0.0);
goal = (0.0, 10.0);

height = -0.01*x + 0.003*x^2 + 0.04 * y;
```

Required API

In order to support automated testing, you are required to implement the following classes and methods. Note that the classes indicated may (and mostly should) have more methods, but you *must* provide at least the methods indicated.

```
class Vector2d {
    public Vector2d(double x, double y);
    public double get_x();
    public double get_y();
}

 interface Function2d {
    public double evaluate(Vector2d p);
    public Vector2d gradient(Vector2d p);
}

class PuttingCourse {
    public PuttingCourse(Function2d height,
                         Vector2d flag, Vector2d start);
    public Function2d get_height();
    public Vector2d get_flag_position();
    public Vector2d get_start_position();
    public double get_friction_coefficient();
    public double get_maximum_velocity();
    public double get_hole_tolerance();
}

interface PhysicsEngine {
}

class EulerSolver : public PhysicsEngine {
    void set_step_size(double h);
}

class PuttingSimulator {
    public PuttingSimulator(PuttingCourse, PhysicsEngine);
    public void set_ball_position(Vector2d);
    public void get_ball_position();
    public void take_shot(Vector2d initial_ball_velocity);
}

interface PuttingBot {
    public Vector2d shot_velocity(PuttingCourse course,
                                  Vector2d ball_position);
}
```