WORKSHEET 2: NEWTONIAN MECHANICS

Version 1.0
Computing
COMP270

Dr Ed Powley

Introduction

In this worksheet you will use Newtonian mechanics to write an auto-aim system for a simple tank game.

Begin by **forking** the following BitBucket repository:

https://gamesgit.falmouth.ac.uk/projects/COMP270/repos/comp270-worksheet-2

Complete the **FOUR** tasks described below, remembering to **commit** your work regularly. To submit your work, open a **pull request** from your forked repository to the original repository.

Important: in this worksheet, you must **only** make edits to the files controller.cpp and controller.h. Any edits to other .cpp or .h files will be **reverted** before your work is marked, regardless of whether this breaks your code! However feel free to add as much code as necessary to controller.cpp and controller.h that makes your solutions easier to write or maintain — for example you are encouraged to add your own helper methods in order to enable code reuse.

Background

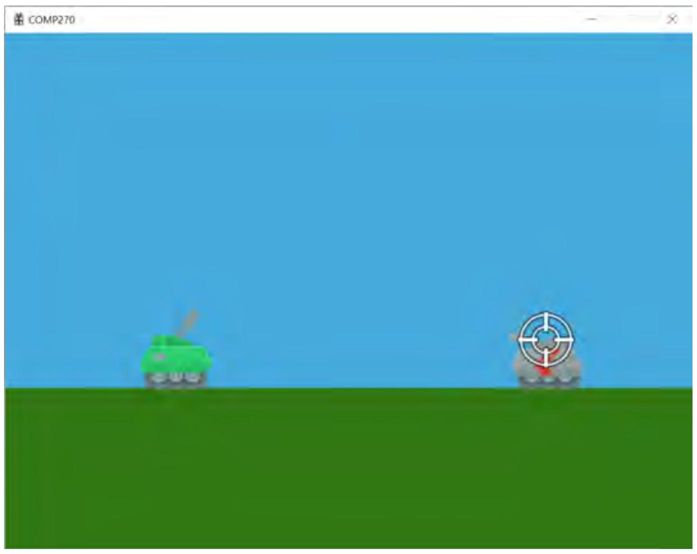

Figure 1: A screenshot of the tank game.

The provided skeleton project implements a simple tank combat game, as shown in Figure 1. The player tank (green, left) and the enemy tank (grey and red, right) spawn in random positions. On pressing the **space bar**, the player tank fires a shell with an initial speed of u pixels per second, at an angle of θ radians from the horizontal (Figure 2).

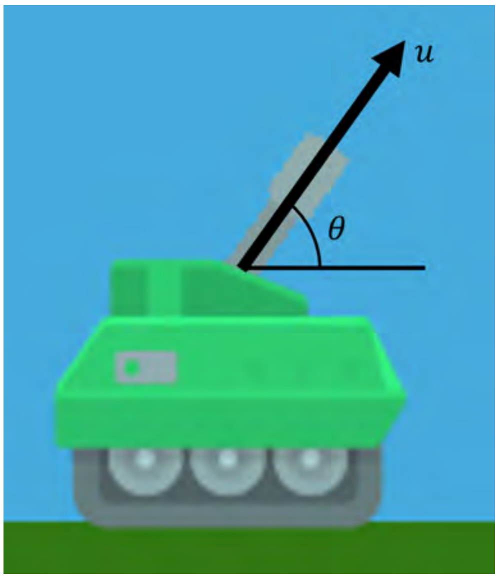

Figure 2: The speed and angle of the shot.

Task 1: calculating the shot speed

When the shot is fired, the program calls `Controller::calculateShotSpeed` to calculate the value of u. The following information is passed in:

- `tankPos`: a vector representing the position of the tank (i.e. the initial position of the bullet)
- `enemyPos`: the target position
- `shotAngleRadians`: the angle θ in Figure 2, which is fixed
- `gravity`: the downward acceleration due to gravity
- `wind`: for now this is always zero and can be ignored

All vectors are represented in screen space, in pixels, with the origin at the top left of the window and the positive y-axis pointing downwards. Speed is represented in pixels per second, and acceleration in pixels per second per second (px/s^2). The program is set up so that `tankPos` and `enemyPos` have the same y component, i.e. `(enemyPos - tankPos).y == 0`.

Modify `Controller::calculateShotSpeed` to calculate the shot speed U required to hit the target. A formula for u was derived in class.

Task 2: accounting for height difference

Edit Controller.h to set `c_canHandleHeightDifference = true`. This removes the restriction that `tankPos` and `enemyPos` have the same y component.

Derive a new formula for u which can handle the situation where `(enemyPos - tankPos).y != 0`. **Modify** your implementation of `Controller::calculateShotSpeed` to use this new formula.

Note that you are **not** asked to provide your full worked derivation as part of your worksheet submission. However make sure you include enough documentation of it in the comments for your code.

Task 3: accounting for wind

Edit Controller.h to set `c_canHandleWind = true`. This adds a constant horizontal acceleration to the bullet, to simulate the effects of wind. This means that the acceleration on the bullet is the vector $\begin{pmatrix} w \\ g \end{pmatrix}$ where w is the acceleration due to wind and g is the acceleration due to gravity. The acceleration due to wind, in px/s^2, is passed into `Controller::calculateShotSpeed` as `wind`.

Derive a new formula for u which can handle wind. **Modify** your implementation of `Controller::calculateShotSpeed` to use this new formula.

Task 4: calculating the shot angle

Edit Controller.h to set `c_doCalculateAngle = true`. Now instead of fixing the shot angle θ and calling `Controller::calculateShotSpeed` to get the shot speed u, the game fixes u and calls `Controller::calculateShotAngle` to get θ.

Derive a formula for θ given u. You may tackle the cases without height difference or wind first (by changing the relevant flags in Controller.h back to `false`), or you may tackle the general case straight away. **Implement** your formula in `Controller::calculateShotAngle`.

Marking Rubric

All submissions and assessment criteria for this assignment are individual. To **pass** this assignment (achieve 40% or more), you must submit a reasonable attempt at the worksheet by the formative deadline stated on LearningSpace.

Criterion	Weight	Near Pass	Adequate	Competent	Very Good	Excellent	Outstanding
Basic competency threshold	30%	A reasonable attempt at the worksheet was not submitted by the formative deadline. Evidence of breach of academic integrity.					
PROCESS: Functional coherence	40%	None of the tasks have been attempted.	Task 1 has been attempted and partially completed.	Task 1 has been successfully completed.	Tasks 1 and 2 have been successfully completed.	Tasks 1–3 have been successfully completed.	Tasks 1–4 have been successfully completed.
PROCESS: Maintainability	30%	The code is only sporadically commented, if at all, or comments are unclear. Few identifier names are clear or inappropriate. Code formatting hinders readability.	The code is well commented. Some identifier names are descriptive and appropriate. An attempt has been made to adhere to a consistent formatting style. There is little obvious duplication of code or of literal values.	The code is reasonably well commented. Most identifier names are descriptive and appropriate. Most code adheres to a sensible formatting style. There is almost no obvious duplication of code or of literal values.	The code is reasonably well commented, with appropriate high-level documentation. Almost all identifier names are descriptive and appropriate. Almost all code adheres to a sensible formatting style. There is no obvious duplication of code or of literal values. Some literal values can be easily "tinkered".	The code is very well commented, with comprehensive appropriate high-level documentation. All identifier names are descriptive and appropriate. All code adheres to a sensible formatting style. There is no obvious duplication of code or of literal values. Most literal values are, where appropriate, easily "tinkered".	The code is commented extremely well, with comprehensive appropriate high-level documentation. All identifier names are descriptive and appropriate. All code adheres to a sensible formatting style. There is no duplication of code or of literal values. Nearly all literal values are, where appropriate, easily "tinkered".