Cuake

a *∅hr* test task.

v 1.0.1

Arsenijs Gluhihs
<arsenijs@doma.dev>

Jons Mostovojs
<jonn@doma.dev>

1 Introduction

Thank you for and agreeing to complete this *∅hr* task! It is designed to take around an hour to complete for a developer proficient in Unity.

You are given a project which has some functions which are yet not implemented. You have to fill in implementations of these functions following the specification given in this document.

Note that there may be bugs in the skeleton implementation. If you find an inconsistency, consider the specification correct and the implementation faulty.

2 The gist

While making computer games, having challenging AIs that perform well is a key to great experience. The authors of Cuake (a game where the player has to use fuel to manage speed and direction of a pair of cubical units in order to ram opponent's cubes) need your help to develop the best AI possible.

At the moment of collision between two opposing units, whichever unit was moving faster will destroy the other one, but its speed will be decreased in line with normal collision physics.

The units are fighting under a cubical glass dome on an icy surface with barriers. The units are perfectly controllable on the ground and in the air, if you manage to lift them off somehow.

All the AI submissions shall enter a Swiss tournament, playing best three out of five matches. Good luck!

3 Setup

You will be given a Unity project with a scene called "MainScene". This is the single integration point for all the assets that makes it possible to launch server and clients via standard "NetworkManager" and play the game itself, via class unsurprisingly called "Game".

There are four spawn points for the pairs of units of both opponents, which are used by "Game" via "GameState" to populate the board when the game starts. The units are represented by cubes, and in the following section, we describe their mechanics.

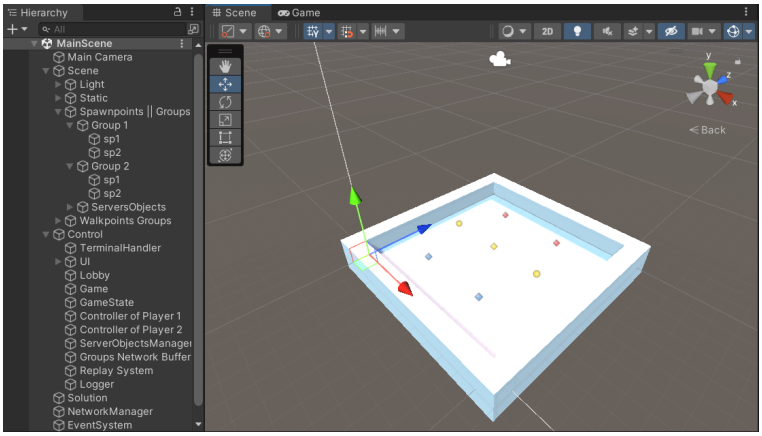

Figure 1: *The red and blue points are spawn points for the units, whereas the yellow ones are the spawn point and the turning points of the trajectory of the neutral cube. (0, 0, 0) is a point, marked by green, blue and red axis.*

"Solution" class has a controller attached to it, using which the cubes are controlled. By default, you have "UniversalUserController" selected in "Solution", which allows a human player to use keyboard to control the units. Pressing "1" and "2" will toggle the selection of own units and "WASD" or arrow keys will move the selected units around. A human player can also hold left mouse button down and drag it in any direction, causing a larger force in that direction be applied to each of the selected units.

4 Cube mechanics

Each player controls two units represented by cubes and spawned at the spawn points on each side of the board (see Fig. 1). The goal of each player is to

collide with opponents' units in such way that their own units travel faster than the opponents' units at collision time. When such collision happens and the difference in speeds is greater than some threshold, a collision event is computed, but the slower cube is removed from the board. The threshold is configured in "GameState" script in the "Removal Speed Threshold" field.

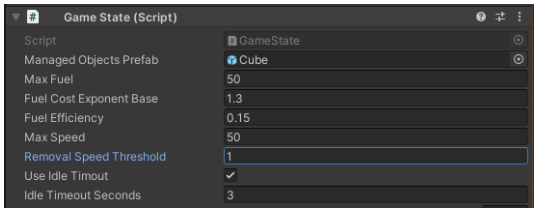

Figure 2: *During collision, the game will access velocity of both colliding cubes and take its magnitude (speed). In this case, it will check that it's greater than* $1\frac{m}{s}$ *and if it is, it will remove the slower cube.*

5 Fuel

The players can't just speed up indefinitely. The speed has a hard higher limit which is normally unreachable.

It is unreachable, because the game has a finite resource: fuel. When "CalculateInput()" method is called each frame by "Player" object, it returns a vector v with a magnitude m. The physical meaning of m is "how much fuel to consume in this frame". Server handling this will calculate an impulse j with magnitude $|j|$.

Let b be "Fuel Cost Exponent Base" and c be "Fuel Efficiency" (see Fig. 2). Then the formula for the magnitude of this impulse is:

$$|j| = c \cdot log_b(m + 1)$$

Impulse j is then applied to each of the currently selected units owned by the player issuing the input and is collinear with v.

6 Unit Selection

As mentioned in the previous section, the impulse is only applied to the units that are selected. You can select an element from anywhere in the code of your

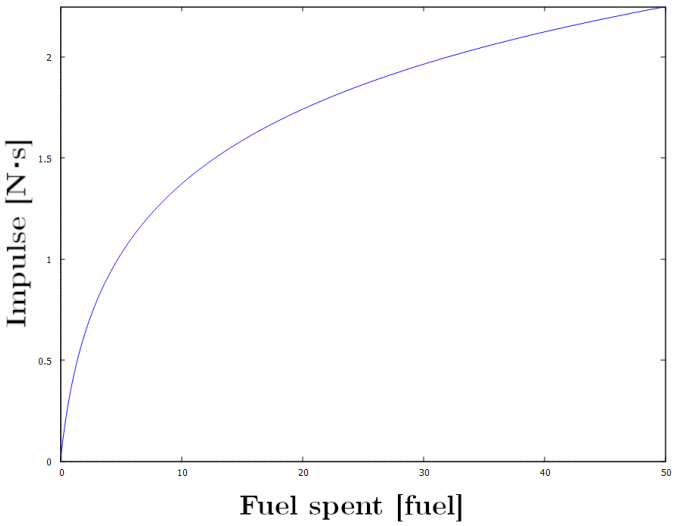

Figure 3: *Impulse generated as a function of fuel spent with game parameters from Fig. 2.*

AI using methods "SelectElement", "DeselectElement" and "ToggleElement".

Note that if you have both units selected and your "CalculateInput()" would produce an input, requesting to spend m fuel, exactly m fuel will be spent, rather than $2m$. And the same impulse will be relayed to both selected elements.

7 The Board

The board consists of the icy surface and bumper walls. On the board there is a moving neutral obstacle called "the walking cube", which doesn't remove the units on collision. Also, on the board there are some units controlled by the players.

7.1 Icy Surface

Icy surface is a 10×10 square on which the cubes spawn. It has a low, but noticeable friction.

To see the physics of this material, consult "PhysicsMaterial/Icy Floor".

7.2 Bumper Walls

Bumper walls are visible from y coordinates $0; 1$ and are completely transparent between y coordinates $1; 11$. The bumper ceiling is also completely transparent.

To see the physics of this material, consult "PhysicMaterial/Bumper Wall".

7.3 The Walking Cube

The Walking Cube spawns somewhere in the middle of the map and then starts traversing the line of delineation. It has a significantly greater mass than normal cubes.

This cube its material with cubes, that represent players' units.

Your AI can get the information about the position and the velocity of this cube by searching the synchronised singleton:

```
ServerObjectsManager.Singleton.ServerObjects
```

7.4 The Units

Each player owns two units, labeled "1" and "2". They have the same physics as "the Walking Cube", except they are significantly lighter.

Your AI can get the information about your units:

```
var myElements =
    GameState.Singleton.Group(MyPlayerId).Elements
```

Whereas you can access the enemy units like this:

```
int enemy = MyPlayerId == 0 ? 1 : 0;
var enemyElements =
    GameState.Singleton.Group(enemy).Elements;
```

8 Submission

Your submission will be competing against a bunch of naive solvers (implemented by us) and against solvers, submitted by other candidates. To qualify for a middle-level position, you're expected to either beat all the naive solvers or to produce an above-average submission. To qualify for a junior position, you're expected to produce a working submission that scores comparable to an average naive solver.

9 Swiss Tournament

The Swiss tournament system is designed to ensure fair competition by pairing participants based on their performance in preceding rounds:

- **Initial Pairing:** In the first round, players are paired randomly.
- **Scoring System:** Points are awarded based on match outcomes as follows:
  - **Win:** 3 points
  - **Draw:** 1 point
  - **Loss:** 0 points
- **Subsequent Pairing:** From the second round onwards, players are paired according to their accumulated points. Those with similar points are matched against each other. This method ensures that as the rounds progress, players of similar strength (based on their performance in the tournament) face each other.
- **Best Three out of Five:** In our setup, each match is a "best three out of five" format. The player to first achieve three game wins out of a possible five is considered the match winner.
- **Match Continuity:** It's crucial for to know that solvers can retain their state between games within a match. This facilitates strategies where a solver might adjust its tactics or compute potential outcomes based on the results of previous games in the match.
- **Tie-breakers:** At times, multiple players might have the same score after the final round. In such cases, we employ the following tie-breaks:
  - **Opponent Match Win Percentage (OMW%):** A metric of cumulative strength of your opponents.
  - **Game Win Percentage (GW%):** A metric of how effective your submission was at winning games, not just matches.
  - **Opponent Game Win Percentage (OGW%):** A metric of how cumulative effectiveness of your opponents' submissions.
  - **Deterministic Coin Flip:** In the extremely unlikely event that all the tie breaks are the same (it's almost impossible), the tie shall be broken by a coin flip.

9.1 Qualifier Tournament

After a submission is received, it enters a qualifier Swiss Tournament, where it's pitted against multiple baseline AIs made by us and the weakest AIs submitted by other candidates. These submissions can be thought of as "naive". This tournament will have 7 other entrants and 3 rounds. For you to qualify for a junior position, your submission has to score at least 6 points in the qualifier.

9.2 Main Tournament

If your submission has managed to pass a qualifier Swiss Tournament, it enters the main Swiss Tournament.

- **Qualification for Middle Position:**
  - Secure 9 points in the Qualifier.
  - Or, have a positive record in the main tournament by scoring more than half of the available points.
- **Qualification for Senior Position:** Achieve a minimum score of $3(R-1)$ points in the main tournament, where R is the total number of rounds.
- **Initial Pairing:** Players are randomly paired up for the first round.

9.3 Victory Conditions, Matches & Games

The AIs are ran once per match of the tournament inside a disposable VM. It means that your submission can preserve the information about the games it played against a given opponent, but will be reset at the beginning of the next round.

Individual games are ran as follows:

- **Preparation Time:** AIs are allocated approximately, but not less than, 100 milliseconds of wall time for game preparations.
- **Game Duration:** AIs have approximately, but not less than, 9 seconds to secure a victory.
- **Victory Conditions:**
  - If an AI successfully removes all of the opponent's cubes within the stipulated time, it is declared the winner immediately.
  - If neither AI achieves this on time, the player with the higher number of cubes remaining is the winner.
  - In the event that both players have an equal count of cubes, the winner is determined by the amount of fuel reserve. The player with the greater reserve is declared the victor.

The AI connects to the server using the functionality defined in "Lobby" via "NetworkManager".