ai cup

CodeSide 2019

Rules

Version 1.0.1

December 2019

Contents

Chapter 1

About CodeSide 2019 world

1.1 General concept of the game and the rules of the tournament

This competition gives you an opportunity to test your programming skills, by creating an artificial intelligence (strategy) controlling a team of units in a special world (you can learn about details of the CodeSide 2019 world in later sections).

In each game you are to compete against another player's strategy. You team's goal — is to gain score by killing or doing damage to your opponent. Team who has more score is the winner. Game can also be tied if both teams have same score in the end.

Time in the game is discrete and is measured in "ticks". At the beginning of each tick, the game simulator transmits the world state data to the participants' strategies, receives actions from them and updates the state of the world in accordance with these actions and the limitations of the world. Then makes calculation of the change of the world and objects in it for this tick, and the process is repeated again with the updated data. The maximum duration of any game is equal to 3600 ticks, but the game can be terminated prematurely if all strategies have "crashed".

The "crashed" strategy can no longer control its units. The strategy is considered as "crashed" in the following cases:

- The process in which the strategy is started has unexpectedly terminated, or an error has occurred in the protocol of interaction between the strategy and game server.
- The strategy exceeded one (any) of the time constraints assigned to it. Strategy for one tick is allocated not more than 1 seconds of real time. But in sum for the whole game the strategy process is given

  $$20 \times <duration\_of\_game\_in\_ticks> + 20000 \tag{1.1}$$

  milliseconds of cpu time. The formula takes into account the maximum duration of the game. The time limit remains the same, even if the actual duration of the game is different from this value. All time limits apply not only to the participant code, but on the interaction of the client-shell strategy with the game simulator.
- The strategy exceeded the memory limit. At any point in time the strategy process should not consume more than 256 MB of RAM.

The tournament is held in several stages preceded by a qualification in the Sandbox. Sandbox is a competition that takes place throughout the championship. The player has a certain rating value – an indicator of how successful their strategy is involved in games within each stage.

The initial value of the rating in the Sandbox is 1200. At the end of the game this value can both be increased and decreased. At the same time victory over a weak (with a low rating) opponent gives a small increase, also the defeat from a strong opponent slightly decreases your rating. Over time the rating in the Sandbox becomes more and more inert, which makes it possible to decrease the impact of random long series of victories or defeats on the participant's place, but at the same time makes it difficult to change their position with a significant improvement in strategy. To cancel such effect the participant can reset the variability of the rating to the initial state when sending a new strategy, including the corresponding option. If the new strategy is adopted, the rating system of the participant will fall dramatically after the next game in the Sandbox, however, further participation in games will quickly recover and even become higher if your strategy has really become more effective. It is not recommended to use this option with minor, incremental improvements to your strategy, as well as in cases where a new strategy insufficiently tested and the effect of changes in it is not known reliably.

The initial value of the rating at each main stage of the tournament is 0. For each game the participant receives a certain number of rating points depending on the occupied place (a system similar to that used in the championship "Formula-1"). If two or more participants share some place, then the total number of rating points for this place and for the following `number_of_such_members` − 1 of places is shared equally among these participants. For example, if two participants share the first place, then each of them will receive half of the rating points number for the first and second places. When sharing rounding always takes place in a smaller direction. More detailed information about the stages of the tournament will be provided in the announcements on the project website.

First all participants can participate only in the games that take place in the Sandbox. Players can send their strategies to the Sandbox, and the last one taken from them is taken by the system for participation in qualifying games. Each player participates in approximately one qualifying game for an hour. The jury reserves the right to change this interval based on the throughput of the testing system, but for the majority of participants it remains constant. There are a number of criteria by which the interval of participation in qualifying games can be increased for a specific player. For every N-th full week that has elapsed since the player sent the last strategy, the interval of participation for this player is increased by N basic test intervals. Only the strategies adopted by the system are taken into account. An additional penalty which is equal to 20% from the basic testing interval is charged in the Sandbox for each strategy "crash" in 10 last games. More details about the causes of the strategy "crashing" can be found in the following sections. The player's participation interval in the Sandbox can not become bigger than a day.

Games in the Sandbox are held according to a set of rules corresponding to the rules of an accidental past stage of the tournament or to the rules of the next (current) stage. At the same time, the closer the rating value of the two players rating within the Sandbox, the more likely that they will be in the one game. The Sandbox starts before the start of the first stage of the tournament and ends after some time after the final stage (see the schedule of stages to clarify the details). In addition, the Sandbox is frozen during the stages of the tournament. Following the results of the games in the Sandbox there is a selection for participation in Round 1, which will involve 1080 of participants with the highest rating at the beginning of this stage of the tournament (if the rating is equal, priority is given to the player who previously sent the latest version of their strategy), as well as an additional selection to the next stages of the tournament, including the Finals.

Tournament stages:

- In **Round 1** you will learn the rules of the game. You are given 1 unit, same as your opponent. The level is pretty simple so no advanced pathfinding is needed. The task is — deal as much damage as you can! It's simple. Round 1, as all further stages, consists of two parts, between which there will be a short break (with the renewal of the Sandbox work), which allows to improve its strategy. The last strategy sent by the player before the beginning of this part is selected for the games in each part. Games are conducted in

waves. In each wave, each player participates exactly in one game. The number of waves in each part is determined by the capabilities of the testing system, but it is guaranteed that it will not be less than ten. 300 highest rated participants will be held in Round 2. Also in Round 2 there will be an additional selection of 60 participants with the highest rating in the Sandbox (at the moment of Round 2 beginning) among those who did not passed according to the results of Round 1.

- In **Round 2** you have to learn teamwork. Now you have 2 units in your team. The task is further complicated that after summarizing the Round 1, the part of the weak strategies will be eliminated and you will have to confront stronger opponents. According to the results of Round 2 of the best 50 strategies will reach the Finals. Also in the Finals there will be an additional selection of 10 participants with the highest rating in the Sandbox (at the beginning of the Finals) from those who did not go through the main tournament.
- **Finals** is the most important stage. After the selection, held following the results of the first two stages, the strongest participants will be remained. Also, the level is going to be more complex now, so more advances pathfinding techniques must be used. The system of holding the Finals has its own peculiarities. The stage is still divided into two parts, but they will no longer consist of waves. In each part of the stage, games will be played between all pairs of Finals participants. If the time and capabilities of the testing system permit, the operation will be repeated.

All finalists are ranked according to the non-increase in the rating after the end of the Finals. If the ratings are equal, a higher place is taken by that finalist, whose strategy, which was part of the Final, was sent out earlier. Prizes for the Final are distributed based on the occupied place after this ordering.

After the completion of the Sandbox, all its participants, except for the Finals winners, are ranked according to the non-increase in the rating. If the ratings are equal a higher place is taken by the participant who sent the latest version of their strategy earlier. Prizes for the Sandbox are distributed on the basis of occupied place after this ordering.

1.2 Description of the game world

The game world is two dimensional. X axis is horizontal, Y axis is vertical, directed up. The level consists of tiles, which have different collision mechanics. All entities (units, bullets, explosions, loot boxes, level tiles) are rectangular, with sides parallel to X and Y axis.

In the beginning of each game, units are placed on their spawn points and have no weapon. Loot boxes spawn on the level containing weapons, health packs and mines.

Game is being updated several times each tick. Each update advances game time by $\frac{1}{ticks\_per\_second \times updated\_per\_tick}$.

1.3 Level

The level is represented by a grid of tiles. Each tile is of size 1×1. Each tile can be one of:

- Empty.
- Wall — blocks all movement and bullets.
- Platform — can be used to stand on, but can be moved through.

- Ladder — can be used to freely move vertically. Unit is considered to be on ladder if the line from center of unit to bottom middle point of unit intersects with the tile.
- Jump pad — gives higher jump. Activated if unit is intersecting with the tile. Jumps off jump pads are not cancellable.

1.4 Movement

Units have no acceleration, and horizontal / vertical movements are independent. Each update, first horizontal movement is performed, then vertical movement.

Horizontal movement speed is set in the action by your strategy. On game update, your unit's X position is changing by velocity set, until there is a tile or other unit blocking movement.

Vertical movement is a little more complicated. Unit can be either in falling, or in jumping state. Jumping can be cancelled any time unless jumped off a jump pad.

If the unit is in falling state, it moves down with constant speed. If falling is blocked by some level tile or other unit, falling stops. Which tiles are blocking depends on whether you set "jump_down" or not. If set, only walls block falling. If not set, platforms and ladders also block. If falling is blocked, unit goes into jumping state. When on ladder, falling state is always changed to jumping state. Jumping state may be reverted immediately if "jump" is not set in your strategy.

If unit is in jumping state, it moves up with constant speed. Jumping state is limited by certain time. After that time state is changed to falling. When jumping, unit's Y position is changing by constant velocity. If jumping is blocked by wall tile or other unit, state is changed to falling. Jumping state can be reverted to falling if "jump" is not set in your strategy.

Jumping time and speed can be different depending on what caused the jump. Jumping can be started manually after hitting ground or by jump pad tiles. In case jump is cause by jump pad, the jump is not cancellable. This means, that jumping state will only be reverted to falling if unit hits ceiling or jump time has passed.

1.5 Looting

When game starts, loot boxes are randomly spawned on the level. A loot box may contain:

- Health pack. Picked up only if unit is not max health currently. Increases health instantly.
- Weapon. Picked up if unit has no weapon equipped currently. It's also possible to swap unit's current weapon with the one in the box.
- Mine. Units have inventory with mines, and they can be placed and activated later.

Loot box can be interacted with if it intersects with the unit.

1.6 Shooting

If unit has a weapon, it can aim and shoot enemies. Each weapon type has following parameters:

- Magazine size. Weapons have infinite ammo, but they should be reloaded once magazine is used.
- Fire rate. Time between shots.
- Reload time.
- Min and max spread. Weapons have current spread value which is between min and max. Bullets shot from the weapon will be shot at angle modified randomly by current spread value.
- Recoil. Current weapon spread value increases by recoil each time a bullet is fired.
- Aim speed. Speed of decreasing weapon's current spread value.
- Bullet parameters — speed, size and damage of a bullet.
- Explosion parameters — may be absent. If present, bullets explode, dealing explosion's damage in given radius.

Bullets are squares. They spawn at the center of unit who made the shot. They hit other units if they intersect with them (but a bullet can't hit the unit who made the shot). Bullets can also hit wall tiles of the level and mines. In any case, bullet disappears immediately, dealing damage if hit a unit. If a mine was hit, it will be exploded. Also, if explosion parameters are present, an explosion is created, dealing damage to all units that intersect with a square with center at the bullet's position and given radius (so, size of explosion's square is twice the radius). Explosions can also hit and explode other mines.

These parameters are constant for each weapon type. There are also varying parameters of a weapon:

- Magazine — ammo left in current magazine.
- Spread — current spread. Initially is equal to min spread. Bullets' angle is modified randomly, uniformly in range $[-spread, spread]$.
- Fire timer — time until next shot. May be absent if shooting is already possible.
- Last angle — last aiming angle. When you change aiming direction, current spread value increases by the angle difference. For the first tick upon picking a weapon, this value is absent.

When weapon is first picked up, it starts with being reloaded. Reloading a weapon also happens automatically if ammo in current magazine reaches zero.

1.7 Mines

Mines can only be planted if unit is currently on ground. Upon planting, some time is required to prepare. During this time the mine can't be triggered by itself. Then mine goes to idle state, waiting for a unit to come close (within its trigger radius). After been triggered, some time will pass and then the mine will explode, dealing damage to all units in specified explosion radius. Explosions can also hit and explode other mines.

1.8 Control

Controlling the unit is done via properly setting up the *Action* returned from your strategy.

By setting *velocity* you control horizontal speed of the unit. The value is clamped to be less than unit's max horizontal speed.

By setting *jump* to *true* you tell the unit to start or continue jumping. If it is set to *false* and current unit's jump is cancellable, the unit stops jumping.

By setting *jump_down* to *true*, unit starts moving down ladders and jumping down from or falling through the platforms.

The *aim* property controls where the unit is aiming. If unit has no weapon, this property does nothing. Set *shoot* to *true* to start shooting.

If *reload* is set to *true*, unit starts reloading its weapon.

Setting *swap_weapon* to *true* will make unit swap its current weapon with the one in the loot box near it, if any.

Finally, *plant_mine* will cause unit to plant a mine in its current position, if unit has mines in inventory.

Chapter 2

API description

2.1 MyStrategy

In language pack for your programming language you can find file named `MyStrategy/my_strategy.<ext>`. This file contains class `MyStrategy` with `get_action` method, where your strategy's logic should be implemented.

This method will be called each tick, separately for each of your units.

The method takes following arguments:

- Current unit for which to compute next action (*Unit* object)
- Current game state (*Game* object)
- Debug helper object (*Debug* object). This object allows you to do custom rendering from inside your strategy code. Note that using this has no effect when testing your strategy on the server, so use it for local debugging only, otherwise it will cause additional time overhead.

The method should return *UnitAction* object, defining the desired action for specified unit.

2.2 Objects description

In this section, some fields may be absent (denoted as $Optional\langle type\rangle$). The way this is implemented depends on the language used. If possible, a dedicated optional (nullable) type would be used, otherwise other methods may be used (like a nullable pointer type).

Some objects may take one of several forms. The way it is implemented depends on the language. If possible, a dedicated sum (algebraic) data type is used, otherwise other methods may be used (like variants being classes inherited from abstract base class).

$float32$ — 32-bit floating point number, is called $float$, and $float64$ is called *double* in some languages.

2.2.1 Bullet

Defines a bullet. Fields:

- $weapon\_type : WeaponType$ — type of the weapon the bullet was shot from
- $unit\_id : int$ — id of unit who shot the bullet
- $player\_id : int$ — id of player of the shooter
- $position : Vec2\langle float64\rangle$ — bullet's position (center of the bullet)
- $velocity : Vec2\langle float64\rangle$ — bullet's velocity (in units per second)
- $damage : int$ — damage that will be dealt to a unit when hit
- $size : float64$ — bullet's size (side length of square that defines the bullet)
- $explosion\_params : Optional\langle ExplosionParams\rangle$ — explosion parameters, if applicable

2.2.2 BulletParams

Parameters used to create a bullet when shooting a weapon. Fields:

- $speed : float64$ — bullet's speed (length of velocity vector)
- $size : float64$ — bullet's size
- $damage : int$ — bullet's damage

2.2.3 Color

Defines color (used for debug rendering). Fields:

- $r : float32$ — red component
- $g : float32$ — green component
- $b : float32$ — blue component
- $a : float32$ — alpha component (opacity)

2.2.4 ColoredVertex

Defines a vertex (used for debug rendering). Fields:

- $position : Vec2\langle float32\rangle$ — vertex position
- $color : Color$ — vertex color

2.2.5 CustomData

Defines custom data sent to *Debug* object (used for debug rendering). May take one of the following forms:

- *Log* — used for logging text. Fields:
  - $text : string$ — text to display

- *Rect* — draw a rectangle. Fields:
  - $pos : Vec2\langle float32\rangle$ — rectangle position (bottom left corner)
  - $size : Vec2\langle float32\rangle$ — rectangle size
  - $color : Color$ — filling color
- *Line* — draw a line segment. Fields:
  - $p1 : Vec2\langle float32\rangle$ — first end point
  - $p2 : Vec2\langle float32\rangle$ — second end point
  - $width : float32$ — line width
  - $color : Color$ — line color
- *Polygon* — draw a convex polygon. Each vertex may be colored separately — color will be interpolated between vertices. Fields:
  - $vertices : List\langle ColoredVertex\rangle$ — list of vertices
- *PlacedText* — text. Fields:
  - $text : string$ — text to display
  - $pos : Vec2\langle float32\rangle$ — position (in game coordinates)
  - $alignment : TextAlignment$ — text alignment
  - $size : float32$ — text font size in pixels
  - $color : Color$ — text color

2.2.6 ExplosionParams

Parameters of explosion from a mine or an exploding bullet. Fields:

- $radius : float64$ — radius of explosion (half of side length of explosion's square area)
- $damage : int$ — damage dealt by explosion

2.2.7 Game

Defines current game state. Fields:

- $current\_tick : int$ — current tick index
- $properties : Properties$ — game's properties (constants)
- $level : Level$ — level (map)
- $players : List\langle Player\rangle$ — list of players (strategies) participating in the game
- $units : List\langle Unit\rangle$ — list of alive units
- $bullets : List\langle Bullet\rangle$ — list of flying bullets
- $mines : List\langle Mine\rangle$ — list of **planted** mines
- $loot\_boxes : List\langle LootBox\rangle$ — list of loot boxes

2.2.8 Item

Defines an item contained in a loot box. May take one of the following forms:

- $HealthPack$ — health pack. Fields:
  - $health : int$ — health restored
- $Weapon$ — a weapon. Fields:
  - $weapon\_type : WeaponType$ — type of the weapon
- $Mine$ — a mine. No fields.

2.2.9 JumpState

Defines unit's jump state. Fields:

- $can\_jump : boolean$ — whether unit can start/continue jumping
- $speed : float64$ — jump speed (in units per second)
- $max\_time : float64$ — max jump time (in seconds)
- $can\_cancel : boolean$ — whether current jump can be canceled

2.2.10 Level

Defines the level. Fields:

- $tiles : List\langle List\langle Tile\rangle\rangle$ — 2d list of level tiles

2.2.11 LootBox

Defines a loot box. Fields:

- $position : Vec2\langle float64\rangle$ — loot box's position (bottom middle point)
- $size : Vec2\langle float64\rangle$ — loot box's size
- $item : Item$ — item contained in this loot box

2.2.12 Mine

Defines a **planted** mine. Fields:

- $player\_id : int$ — id of player whose unit planted the mine
- $position : Vec2\langle float64\rangle$ — mine's position (bottom middle point)
- $size : Vec2\langle float64\rangle$ — mine's size
- $state : MineState$ — current mine state

- $timer : Optional\langle float64\rangle$ — time left until state change. May be absent
- $trigger\_radius : float64$ — mine's triggering radius
- $explosion\_params : ExplosionParams$ — explosion params

2.2.13 MineState

Defines mine's state. Variants:

- Preparing
- Idle
- Triggered

2.2.14 Player

Defines player (strategy) participating in the game. Fields:

- $id : int$ — player's id
- $score : int$ — current player's score

2.2.15 Properties

Defines game's properties (constants). Fields:

- $max\_tick\_count : int$ — max game duration in ticks
- $ticks\_per\_second : float64$ — number of ticks per "second"
- $updates\_per\_tick : int$ — number of updates per tick. Each update advances game time by $\frac{1}{ticks\_per\_second \times updates\_per\_tick}$
- $loot\_box\_size : Vec2\langle float64\rangle$ — size of all loot boxes
- $unit\_size : Vec2\langle float64\rangle$ — size of all units
- $unit\_max\_horizontal\_speed : float64$ — max horizontal speed of a unit
- $unit\_fall\_speed : float64$ — unit's fall speed
- $unit\_jump\_time : float64$ — unit's regular jump time
- $unit\_jump\_speed : float64$ — unit's regular jump speed
- $jump\_pad\_jump\_time : float64$ — unit's jump time off a jump pad
- $jump\_pad\_jump\_speed : float64$ — unit's jump speed off a jump pad
- $unit\_max\_health : int$ — max (starting) unit's health
- $weapon\_params : Map\langle WeaponType \to WeaponParams\rangle$ — weapon parameters by weapon type
- $mine\_size : Vec2\langle float64\rangle$ — size of **planted** mines

- $mine\_explosion\_params : ExplosionParams$ — explosion params for mines
- $mine\_prepare\_time : float64$ — mine preparing time
- $mine\_trigger\_time : float64$ — mine triggering time
- $mine\_trigger\_radius : float64$ — mine triggering radius
- $kill\_score : int$ — score given for each unit killed

2.2.16 TextAlignment

Defines text alignment (used for debug rendering). Variants:

- Left
- Center
- Right

2.2.17 Tile

Defines level's tile. Variants:

- Empty
- Wall
- Platform
- Ladder
- JumpPad

2.2.18 Unit

Defines a unit. Fields:

- $player\_id : int$ — owner player's id
- $id : int$ — unit's id
- $health : int$ — unit's health
- $position : Vec2\langle float64\rangle$ — unit's position (bottom middle point)
- $size : Vec2\langle float64\rangle$ — unit's size
- $jump\_state : JumpState$ — current unit's jump state
- $mines : int$ — number of mines in unit's inventory
- $weapon : Optional\langle Weapon\rangle$ — current unit's weapon, if any

2.2.19 UnitAction

Defines unit's action. Fields:

- $velocity : float64$ — target horizontal velocity
- $jump : boolean$ — whether to start/continue jumping or go up ladders
- $jump\_down : boolean$ — whether to jump down through platforms/go down ladders
- $aim : Vec2\langle float64\rangle$ — aiming direction. Ignored if length is less than 0.5
- $shoot : boolean$ — controls shooting/reloading
- $reload : boolean$ — whether to reload your weapon
- $swap\_weapon : boolean$ — whether to swap weapon with the one in a loot box
- $plant\_mine : boolean$ — controls planting mines

2.2.20 Vec2

Defines a 2d vector. Fields:

- $x : float$
- $y : float$

2.2.21 Weapon

Defines unit's weapon. Fields:

- $type : WeaponType$ — weapon's type
- $params : WeaponParams$ — weapon parameters
- $magazine : int$ — current magazine size
- $spread : float64$ — current spread value
- $fire\_timer : Optional\langle float64\rangle$ — time until next possible shot (absent if shooting is possible already)
- $last\_angle : Optional\langle float64\rangle$ — last aiming angle (absent upon pickup)

2.2.22 WeaponParams

Defines weapon parameters (constants). Fields:

- $magazine\_size : int$ — max magazine size
- $fire\_rate : float64$ — time between consequent shots
- $reload\_time : float64$ — time required for reloading
- $min\_spread : float64$ — min spread value

- $max\_spread : float64$ — max spread value
- $recoil : float64$ — recoil value
- $aim\_speed : float64$ — aiming speed
- $bullet : BulletParams$ — bullet parameters
- $explosion : Optional\langle ExplosionParams\rangle$ — bullet's explosion parameters (if applicable)

2.2.23 WeaponType

Variants:

- Pistol
- AssaultRifle
- RocketLauncher