Tank game

Björn Weström <bjorn.westrom@consoden.se>

1 Introduction

Your team is the crew on a battle tank of model Panzer IV. You are currently located in the front of an unnamed warzone, and you radar has just picked up an enemy tank in you neighborhood!

The generals have laid down rules of engagement that must be followed to the point by all troops, including enemy troops. One important such rule is that tanks may only move in four different directions (Left, Right, Up or Down). A similar rule applies to firing missiles; these may only be fired in the above mentioned directions.

The generals have recently acquired a new BMS (Battle Management System), but to their great demise the BMS suffers from some severe limitations. One such limitation is the frequency of updates. Accordingly, the generals made an addition to the rules that all troops, including tanks, must follow one direction of movement during a predefined period of time (called a round) before they are allowed to change direction. Otherwise the generals would not be able to see the status of the battle clearly, and this cannot be allowed! A positive side effect of this rule is that the position of every troop can be described as a pair of integers (x,y), called a square.

Secondly, the BMS (and the generals) can only handle one missile per tank at any time. Hence, the generals added a rule that each tank may only have one missile on the game board at any time.

Due to a very tight time schedule (not to mention budget), the developers of the BMS has chosen to lock down the size of one battle to fixed numbers (width x height), called a game board. Do note that these numbers are not permanently fixed, but may vary from game board to game board. Due to screen resolution limitations on the generals laptops, the width will never exceed 14, and the height will never exceed 12.

Now, since the generals didn't appreciate that the troops might escape from the game board by moving outside the borders, they ordered the engineering troops to install teleport stations around the whole game board just before each battle. This means that a tank in square (9,6) on a game board with width 10 (allowed x coordinates 0-9) that moves to the right will end up in square (0,6). However, the teleport stations does not work for missiles, as the generals concluded that it would be "unfair play" to allow missiles to loop the game board forever.

Your tank is an upgraded model with automatic mineplacer and a large missile ramp. The automatic mineplacer places a mine in every square the tank leaves. The large missile ramp can fire missiles of model Giant Bazinga, which are twice the size of the tank and twice as fast! None of the tank designers have been able to give a trustworthy explanation to how this is technically possible...

Since your tank has already been in service for quite some time, it has one minor defect - it cannot stop! Hence, you must keep moving every round.

The game board consists of two tanks (your own and the enemy), and some walls. As the game moves on, it will also contain more and more mines that the tanks place out along their ways. Tanks and walls are really hard objects, so moving into a wall will destroy your tank and moving into the other tank will destroy both tanks. Mines and missiles are highly explosive, so moving into a mine, or being hit by a missile, will cause your tank to explode too.

2 Entities

To describe how the players move, and how the game board looks, there are a few entities defined. Some of the entities contain additional members to those described below. We have intentionally left out parts that are of no use to the players, so all such members can be ignored. *This does not mean that you will need to use all the information below in your logic, use what makes sense for your team.*

2.1 GameState

The GameState entity describes the game board and everything on it. The game engine process (tank_engine) updates the game state once per second once it has evaluated how everything moves and if anything is hit.

Name	Type	Description
Width	Int32	The fixed width of the game board. Valid x coordinate positions are in the range 0 - (Width-1). Moving left means reducing the x coordinate while moving right means increasing the x coordinate (note that this is reversed if a tank wraps the game board).
Height	Int32	The fixed height of the game board. Valid y coordinate positions are in the range 0 - (Height-1). Moving up means reducing the y coordinate while moving down means increasing the y coordinate (note that this is reversed if a tank wraps the game board).
Tanks	Array	Array of Tank entities (see below). This describes the state of each tank. During the games, only the first two array positions are populated, the rest are set to null.
Missiles	Array	Array of Missile entities (see below). This describes the state of each missile. Active missiles may exist anywhere in the array, inactive missiles (exploded or left the game board) are set to null.
Board	Binary	String of characters describing the game board, indicating where walls and mines are located. A helper class is provided in the example code that parses this string.

2.1.1 Tank

Subentity to the GameState, describing the tank state.

Name	Type	Description
TankId	Int32	The Id code for the tank. Used to determine which joystick controls which tank.
PlayerId	InstanceId	Used to determine which player owns a tank.
PosX	Int32	x position of the tank.
PosY	Int32	y position of the tank.
Fire	Boolean	Indicates if the tank is firing a missile.
MoveDirection	Direction	Indicates the direction of movement for the tank.
TowerDirection	Direction	Indicates the missile tower direction (firing direction) of the tank.
InFlames	Boolean	Indicates if the tank has exploded (it is in flames). An exploded tank is destroyed and cannot make any further moves.

2.1.2 Missile

Subentity to the GameState, describing the missile state.

Name	Type	Description
TankId	Int32	The Id code for the tank that fired the missile.
HeadPosX	Int32	x position of the head of the missile. (Missiles occupy two squares)
HeadPosY	Int32	y position of the head of the missile. (Missiles occupy two squares)
TailPosX	Int32	x position of the tail of the missile.
TailPosY	Int32	y position of the tail of the missile.
Direction	Direction	Indicates the direction of movement for the missile.
InFlames	Boolean	Indicates if the missile has exploded (it is in flames). An exploded missile is destroyed and cannot make any further moves. It will disappear next round.

2.2 Joystick

The joystick entity is used by the player to indicate how it wants to move next round. Note that all movements are subject to the game rules - if the joystick is set to an invalid move (such as MoveDirection null), then this is ignored and overruled by the rules (MoveDirection null is interpreted as Left).

Name	Type	Description
GameId	InstanceId	The Id code of the game that the joystick participates in.
PlayerId	InstanceId	The Id code of the player that owns the joystick.
TankId	Int32	The Id code of the tank the joystick controls.
MoveDirection	Direction	Indicates the requested direction of movement for the tank.
TowerDirection	Direction	Indicates the missile tower direction (firing direction) for the tank.
Fire	Boolean	Indicates if a missile should be fired next round.

2.3 Direction

The direction entity is an enumeration of valid directions. It contains only the values (Left, Right, Up, Down).

2.4 Player

The player entity is used to communicate the name of the player. This is already included in the example code, each team should only change the name string to their team name (see the Virtual Machine Guide document).

3 Rules

The main rules have already been outlined in the introduction, but to clarify the fine points of the rules, the complete set of rules are summarized below.

3.1 General rules

- The game is round based, with a fixed time for each round determined by the game engine. The time per round is 1 second.
- Each player may set their joysticks as many times as they like during each round. The game engine will only read out the current joystick state for the players at the end of each round, and then evaluate the game state for next round. Once the new game state is ready, it is published to the players by an Update event of the GameState entity.
- Each player program will be locked down to one CPU core, so one player program cannot ”steal” CPU cycles from its opponent.
- Each player controls one tank. The game is over once one or both tanks are destroyed.

3.2 Tank rules

- Tanks occupy one square.
- Tanks move one square per round.
- Tanks may move in directions Left, Right, Up or Down.
- Tanks may change direction at any time without time penalty.
- Tanks must move every round. If the player sets the joystick MoveDirection to Null, the game will default to move the tank to the left.
- During each move, both tanks drops a mine in their previous square
- Moving into a mine causes the tank to explode
- Moving into a wall square causes the tank to explode
- Moving into a square where a missile enters during next round causes the tank to explode.

- The special case where a tank is just in front of a missile, and moves against the direction of the missile, will not save the tank despite the fact that the new position does not contain a missile next round. The tank will still explode, staying in its original position.
- Moving into enemy tank causes both tanks to explode (draw game)
- Tanks may move outside the border of the game board, the tank is then wrapped to the opposite side of the game board. Example: Game board is 10x10 squares. Tank is at position (0,0) and moves up. Tank ends up in position (0,9).

3.3 Missile rules

- Missiles normally occupy two squares. The only exception is when a missile hits a wall. If both missile squares would end up inside the wall, the missile is compacted into a single square explosion at the outer wall square - hence a missile cannot reach through single square walls.
- Missiles move two squares per round.
- Missiles may be fired in directions Left, Right, Up or Down.
- Missiles may be fired at any time, but only one missile per tank may be active (exist on the game board) at any time. Trying to fire while again before the last shot missile has disappeared will have no effect.
- A newly fired missiles position is based on the tanks next position (after the mandatory move) and the tower direction. The missile will occupy the two squares closest to the tanks next position in the firing direction. Example: Tank in (1,1), moving right. Its next position is (2,1). If firing direction is down, the missile will occupy (2,2) and (2,3). If firing direction is right, the missile will occupy (3,1) and (4,1).
- A missile that hits a wall explodes. Next round it will be gone and a new missile may be fired.
- A missile that hits a tank explodes. Next round it will be gone, but the game is already finished.
- A missile that completely leaves the game board (both squares outside game board) is considered gone and a new missile may be fired. Missiles do not wrap like tanks do.
- Tower direction need only to be set in the round when the missile is fired, there is no time penalty to change tower direction.
- Changing tower direction while the missile is active in the game board has no effect on the missile.
- Firing with TowerDirection set to Null has no effect.
- If the tank is destroyed in the next round, a missile may still be fired unless the tank moved into a wall square.

3.4 Match game rules

- Every pair of players meet each other in group play. The four best implementations will meet in a final tournament combat.
- Every match consists of three game boards and every game board is played twice with opposite starting positions to ensure that no player gets an advantage from its starting positions. You may not know these game boards in advance!
- A won match gives the player 3 points.
- A draw match gives the player 1 point.
- After the group play is finished, the points for each player is added up to a total score. The four players with highest total scores qualify for the finals.
- In case more than four teams qualify due to equal total scores, additional matches will be held between those players to determine the finalists.
- In the finals, two semi finals with four game boards are played first. The winning players then meet in the final with five game boards.

4 Tools

Each team may use up to three computers, one per team member. The teams may bring any written documentation they whish. The teams may use the Internet to access reference material on C++, C#, Java or Safir SDK Core. No other types of Internet activity is allowed, and it is absolutely not allowed to download or copy code found online or brought to the contest on USB sticks or other movable media. If any suspicios activity is detected we reserve the right to disqualify teams from the contest.

5 Player code skeleton, C++

The code skeleton provided to the teams is a fully functionaly player, although a rather poor one. All communication to the other game processes is already included, and the team can focus on the games strategy and how to implement it. Here is a brief description of the classes provided in the code skeleton.

To simplify handling of coordinates, std::pair is used to encapsulate x and y coordinates in one object. There are many examples in the code of how to use it.

5.1 TankLogic

Main logic class for the player.

5.1.1 MakeMove

This function is called every time the GameState entity is updated, and also when the GameState appears for the first time in a new game. Here you need to place the code to determine what move your player should make. Before returning, call the SetJoystick function (see below) to signal your action to the game engine.

5.1.2 SetJoystick

This function sets the joystick entity to the requested MoveDirection, TowerDirection and Fire state.

5.2 GameMap

Helper class for TankLogic that does parsing of the GameState entity. In the version provided in the skeleton, it does not reuse any data from previous moves, so it should be created new every time MakeMove in TankLogic is called (see code skeleton).

Do note that some of the functions in this class are not intended to be used as is in strong game strategies, but are rather provided as code examples on how to access the neccessary information about the game board, tanks and missiles.

5.2.1 IsEmpty

Returns wheter the argument square is empty (contains no wall and no mine). Note that it may contain missiles or tanks.

5.2.2 OwnPosition

Returns the current position of the players tank.

5.2.3 EnemyPosition

Returns the current position of the enemy tank.

5.2.4 IsMissileInPosition

Returns wheter a missile is in the argument square.

5.2.5 PrintMap

For debugging assistance, prints out the parsed game board (only walls and missiles).

5.2.6 PrintState

For debugging assistance, prints out the parsed game state (player position, enemy position, game board and missiles).

5.2.7 MoveLeft, MoveRight, MoveUp, MoveDown

Convenience functions, returns the position one step (to the left, to the right, up and down respectively) of the argument position. Takes wrapping into account.

5.2.8 SizeX

Returns the x coordinate size of the game board. Valid positions are in the range 0 - (SizeX - 1).

5.2.9 SizeY

Returns the y coordinate size of the game board. Valid positions are in the range 0 - (SizeY - 1).

5.2.10 Elapsed

Returns number of milliseconds elapsed since this GameMap was created.

5.2.11 Index

Private helper functions that return the index of a position in the board string of the GameState.

5.3 Player

This class handles all the communication with the game framework. You should not need to edit this class. If you feel that you do, contact the game judges first.

6 Player code skeleton, C#

The code skeleton provided to the teams is a fully functionaly player, although a rather poor one. All communication to the other game processes is already included, and the team can focus on the games strategy and how to implement it. Here is a brief description of the classes provided in the code skeleton.

To simplify handling of coordinates, a struct Position is declared in GameMap.cs, consisting of two integers (x and y). There are many examples in the code of how to use it.

6.1 TankLogic

Main logic class for the player.

6.1.1 MakeMove

This function is called every time the GameState entity is updated, and also when the GameState appears for the first time in a new game. Here you need to place the code to determine what move your player should make. Before returning, call the setJoystick function (see below) to signal your action to the game engine.

6.1.2 setJoystick

This function sets the joystick entity to the requested MoveDirection, TowerDirection and Fire state.

6.2 GameMap

Helper class for TankLogic that does parsing of the GameState entity. In the version provided in the skeleton, it does not reuse any data from previous moves, so it should be created new every time MakeMove in TankLogic is called (see code skeleton).

Do note that some of the functions in this class are not intended to be used as is in strong game strategies, but are rather provided as code examples on how to access the neccessary information about the game board, tanks and missiles.

6.2.1 IsEmpty

Returns wheter the argument square is empty (contains no wall and no mine). Note that it may contain missiles or tanks.

6.2.2 OwnPosition

Returns the current position of the players tank.

6.2.3 EnemyPosition

Returns the current position of the enemy tank.

6.2.4 IsMissileInPosition

Returns wheter a missile is in the argument square.

6.2.5 PrintMap

For debugging assistance, prints out the parsed game board (only walls and missiles).

6.2.6 PrintState

For debugging assistance, prints out the parsed game state (player position, enemy position, game board and missiles).

6.2.7 MoveLeft, MoveRight, MoveUp, MoveDown

Convenience functions, returns the position one step (to the left, to the right, up and down respectively) of the argument position. Takes wrapping into account.

6.2.8 SizeX

Returns the x coordinate size of the game board. Valid positions are in the range 0 - (SizeX - 1).

6.2.9 SizeY

Returns the y coordinate size of the game board. Valid positions are in the range 0 - (SizeY - 1).

6.2.10 Elapsed

Returns number of milliseconds elapsed since this GameMap was created.

6.2.11 ToIndex

Private helper functions that return the index of a position in the board string of the GameState.

6.3 Player

This class handles all the communication with the game framework. You should not need to edit this class. If you feel that you do, contact the game judges first.

7 Player code skeleton, Java

The code skeleton provided to the teams is a fully functionaly player, although a rather poor one. All communication to the other game processes is already included, and the team can focus on the games strategy and how to implement it. Here is a brief description of the classes provided in the code skeleton.

To simplify handling of coordinates, a class Position is declared in GameMap.java, consisting of two integers (x and y). There are many examples in the code of how to use it.

7.1 TankLogic

Main logic class for the player.

7.1.1 makeMove

This function is called every time the GameState entity is updated, and also when the GameState appears for the first time in a new game. Here you need to place the code to determine what move your player should make. Before returning, call the SetJoystick function (see below) to signal your action to the game engine.

7.1.2 setJoystick

This function sets the joystick entity to the requested MoveDirection, TowerDirection and Fire state.

7.2 GameMap

Helper class for TankLogic that does parsing of the GameState entity. In the version provided in the skeleton, it does not reuse any data from previous moves, so it should be created new every time MakeMove in TankLogic is called (see code skeleton).

Do note that some of the functions in this class are not intended to be used as is in strong game strategies, but are rather provided as code examples on how to access the neccessary information about the game board, tanks and missiles.

7.2.1 isEmpty

Returns wheter the argument square is empty (contains no wall and no mine). Note that it may contain missiles or tanks.

7.2.2 getOwnPosition

Returns the current position of the players tank.

7.2.3 getEnemyPosition

Returns the current position of the enemy tank.

7.2.4 isMissileInPosition

Returns wheter a missile is in the argument square.

7.2.5 printMap

For debugging assistance, prints out the parsed game board (only walls and missiles).

7.2.6 printState

For debugging assistance, prints out the parsed game state (player position, enemy position, game board and missiles).

7.2.7 moveLeft, moveRight, moveUp, moveDown

Convenience functions, returns the position one step (to the left, to the right, up and down respectively) of the argument position. Takes wrapping into account.

7.2.8 getSizeX

Returns the x coordinate size of the game board. Valid positions are in the range 0 - (SizeX - 1).

7.2.9 getSizeY

Returns the y coordinate size of the game board. Valid positions are in the range 0 - (SizeY - 1).

7.2.10 getElapsedTime

Returns number of milliseconds elapsed since this GameMap was created.

7.3 Player

This class handles all the communication with the game framework. You should not need to edit this class. If you feel that you do, contact the game judges first.