Tron 3D

Albert Vaca Jordi Petit

May 22, 2017

1 Game rules

In this game 2 or 4 players will compete at the same time. Each player will command several Tron-like bikes, that will spawn in predefined spots of the board.

There are several boards where the games can be played. Each board is just a graph made of connected vertices. The cool part is that those graphs have the topology of 3D models, so the game can be represented in awesome 3D graphics!

At each round, players must move their bikes. When bikes move, they leave a solid trail behind them. Bikes that crash into those trails will explode, reducing the score of the player who commands them. Bikes that just do not move for one round will overheat and explode, too! When a bike explodes, its trail will disappear.

The goal of the game is keeping your bikes alive for as long as possible, while trying to make the others crash. The players' score is calculated based in the number of movements their bikes did. Each movement is worth 10 points so the more rounds you keep your bikes alive (and thus moving), the greater your score will be.

When, each round, the players command their bikes, they have no way to know the movements of the other players for that same round. The execution order of the actions of the different players is chosen at random, so if two or more bikes try to move to the same vertex, one at random will be allowed to move, while the others will crash!

1.1 Bonus

To keep the game interesting, we have bonus items too!

In a given round, bonus items will spawn at some vertices of the board (if those are not occupied by bikes or their trails). The bikes that pick up those items by moving to those vertices and will be able to use them to gain super-powers. Note that, until used, items are kept per bike and not per player, and

that a bike can only have a single item at a time, i.e. picking up a second item will replace the previous one. The following bonus items exist:

- Turbo: When activated, it allows the bike to move in turbo mode for some rounds. Turbo mode is explained later on this document.
- Ghost: When activated, it allows the bike to go through walls (but not bikes) for some rounds.
- Extra points: Increases the score of the player by a small amount. This item is consumed automatically upon picking it up. Thus, it will not replace others items the bike could have.

Turbo mode

To implement turbo mode without adding extra complexity, the following decision was taken: The total game duration is doubled, but the normal (non-turbo) bikes are only allowed to move in even rounds, while the turbo bikes can move in both odd and even rounds (thus doubling their move speed). Okay, we know it's not the most elegant solution on the Earth, but the monkeys we have as interns were not able to find a better one. When writing your strategy, take into account that most of the times you will only be able to move half of the rounds! Also note that it is mandatory for bikes in turbo mode to be moved those extra rounds or they will explode too!

Using the turbo can be dangerous because the bike has to move the extra rounds it is given. However, those extra moves will also give the player movement points.

1.2 Game parameters

A game is defined by a given graph (the game board) and the following set of parameters:

- *nb_players*: Number of teams in the game (will be 2 or 4).
- *nb_bikes*: Number of bikes per player (usually 2).
- *nb_rounds*: Number of rounds that will be played (usually 200, that means 100 without turbo).
- *bonus_round*: The round where the bonus items will appear (usually 50).
- *turbo_duration*: The movements a bike can perform in turbo mode after using a turbo item (usually 8, that means 4 extra movements).
- *ghost_duration*: The movements a bike can perform in ghost mode after using the ghost item (usually 3).
- *score_bonus*: The points given when picking up an extra points item (usually 50).

All these parameters can be accessed by the players during the game.

Each board also defines a starting point for each bike and a set of vertices where the bonus items will appear, intended to be fair. To know which are those vertices, the players can check *bonus_vertices* and try to get there before their rivals, but note that if these vertices are already occupied by a wall or a bike, the bonus will not appear.

2 Programming

The first thing you should do is to download the source code.

The source code includes a C++ program that runs the games and an HTML5/Javascript viewer to watch them in a nice animated format. Also, a "Demo" player is provided to make it easier to start coding your own player.

2.1 Running your first game

Here we will explain how to run the game under Linux, but it should work under Windows, Mac, FreeBSD, OpenSolaris... You will only need `g++` and `make` installed on your system, plus a modern browser like Mozilla Firefox or Chromium.

1. Open a console and `cd` to the directory where you extracted the source code.
2. Run `make all` to build the game and all the players. Note that the Makefile will identify as a player any file matching the expression "`AI*.cc`".
3. Make should create an executable file called `Game`. This executable allows you to run a game using a command like:

   ```
   ./Game Demo Demo Demo Demo < icosahedron.cnf > icosahedron.t3d
   ```

   Here, we are starting a match with 4 instances of the player "Demo" (included with the source code), in the board defined in "`icosahedron.cnf`". The output of this match will be stored in "`icosahedron.t3d`".
4. To watch the game, open the viewer (`viewer.html`) with your browser and load the "`icosahedron.t3d`" file.

Use `--help` to see a list of parameters you can use. Particularly useful is `--list`, that will show a list with all the recognized player names.

If needed, remember you can run `make clean` to delete the executable and object files and start over the build.

2.2 Adding your player

To create a player copy the file `AINull.cc` or `AIDemo.cc` to a new file with the same name format (`AIWhatever.cc`).

Then, edit the file you just created and change the playername line to your own player name, as follows:

#define PLAYER_NAME Whatever

The name you choose for your player must be unique, non-offensive and less than 12 letters long. This name will be shown in the website and in the matches.

Then you can start implementing the virtual method *play* (), inherited from the base class *Player*. This method will be called every round and is where your player should decide what to do, and do it.

Of course, you can define auxiliary methods and variables inside your player class, but the entry point of your code will always be this *play* () method.

From your player class you can also call functions to access the board state (defined in the *Board* class in `Board.hh`) and to command your units (defined in the *Action* class in `Action.hh`). Those functions are made available to your code using multiple inheritance, but do not tell your Software Engineering teachers because they might not like it. The documentation about the available functions can be found in the header files of each class. If you have Doxygen installed on your syste, you can also generate a separate document with all the documentation running "make doxygen".

Note that you should not modify the *factory* () method from your player class, nor the last line that adds your player to the list of available players.

2.3 Playing against the Dummy player

To test your strategy against the Dummy player, we provide precompiled `AIDummy.o` object files. This way you still won't have the source code of our Dummy strategy, but you will be able to add it as a player and compete against it locally.

To add the Dummy player to the list of registered players, you will have to edit the `Makefile` file and uncomment one of the first lines matching your platform (Linux 32 bits, Linux 64 bits, MacOS...). Remember that object files contain binary instructions targeting a specific machine, so we can not provide a single, generic file. If you miss an object file for your architecture, contact us and we will try to add it.

Pro tip: You can ask your friends for the object files of their players and add them to the `Makefile` too!

2.4 Restrictions when submitting your player

When you think your player is strong enough to enter the competition, you should submit it to the Jutge website. Since it will run in a secure environment to prevent cheating, some restrictions apply to your code:

- All your source code must be in a single file (`AIWhatever.cc`).
- Your code cannot use global variables (use attributes in your class instead).

- You are only allowed to use standard libraries like `vector`, `map`, `cmath`...
- Your code cannot open files nor do any other system calls (threads, forks...).
- Your CPU time and memory usage will be limited when executed on the server. The time limit are 2 seconds for the execution of the entire game. When watching a game evaluated by the game judge, you will be able to see the percent of the total time limit that your program has used.
- Your program should not write to **cout** nor read from **cin**. You can write debug information to **cerr**, but remember that doing so on the code you upload can waste part of your limited CPU time.

3 Tips

- Read the headers of the classes you are going to use. Do not worry about the private parts nor the implementation.
- Start with simple strategies, easy to code and debug, since this is exactly what you will need at the beginning.
- Before competing with your classmates, focus on defeating the "Dummy" player.
- Try to keep your code clean, it will be easier to change it and to add new behaviours to your strategy.
- Define simple (but useful) auxiliary methods, and *make sure they work properly*.
- Keep a copy of the old versions of your player. When you try to improve it, make it fight against its previous incarnations to measure the improvement.
- As always compile and test your code often. It is *much* easier to trace a bug when you only have changed few lines of code.
- Use `cerr` to output debug information and add `asserts` to make sure the code is doing what it should do. Remember to remove the `cerrs` before uploading your code, because it makes the execution slower.
- When debuging a player, remove the `cerr` you may have in the other player's code, to make sure you only see the messages you want.
- If using `cerr` is not enough to debug your code, learn how to use `valgrind`, `gdb` or any other debugging tool, they are quite useful!
- Switch on the `DEBUG` option in the Makefile, it will allow you to get useful backtraces when your program crashes. There is also a `PROFILE` option you can use.
- Make sure your program is fast enough, the CPU time you are allowed to use is rather short.
- Try to figure out the strategies of the other players by watching the games. This way you could try to defend against them or even improve them in your own player.
- Do not give your code to anybody. Not even an old version. We are using plagiarism detectors to check the programs, not only between them but also against other years submissions.
- You could, however, share the compiled `.o` files or (the easy way) just use the website to play against your friends.

- You can submit new versions of your program at any time.
- Do not wait to the last minute to submit your player. When there are lots of submisions at the same time, it will take longer for the server to run the games, and maybe it is too late!
- Most of the game parameters (number of rounds, duration of bonuses...) won't change, but if your strategy can adjust to them, you will be extra-safe in case we need to change some of them.
- If you create your own map for the game, send it to us before the competition starts and maybe we will include it! The maps should be OBJ models with a single mesh, made of triangles and/or quads, and have equidistant starting points for the bikes. In case you want to change some game parameters for your map (e.g.: ultra-long turbos, only one bike per player...), the previous point will apply :)
- And again: Keep your code simple, build often, test often. Or you will regret.