The Purge

Albert Oliveras

November 23, 2020

1 Game Rules

In order to reduce the number of crimes in the country, the political regime called *The New Founding Fathers of America* has designed a cathartic measure: *The Annual Purge*. Every year, for a number of days, crimes can be committed at night without facing justice for them. The country has been organized in clans of citizens that try to survive.

This is a game for four players, identified by numbers from 0 to 3. Each player has control over a clan made up of warriors and builders. Each of them, which we will call in general "citizens", is born with certain number of points of life.

The game lasts a certain number of days. Each day begins with R daylight rounds, during which crimes are not allowed, and continues with R night rounds, during which the crimes are allowed. There are elements of the game whose behaviour depends on whether it is night or day. In each round, every member of a clan can perform at most one action. As a result of these actions, clans can win points. More precisely, points are obtained by killing rival citizens or collecting money. The winner of the game is the clan that gets the most points.

The game board is an $N \times M$ rectangle from which no one can get out, which represents a city. A position in the board is defined by a pair of integers (r, c) with $0 \leq r < N$ and $0 \leq c < M$. The top-left position is $(0, 0)$ and the bottom-right position is $(N - 1, M - 1)$. Each cell is either part of a street or a building. Citizens cannot step on any building and must move through the streets. In the streets, one can find weapons, money and food. In order to avoid possible attacks, barricades can be built.

Barricades.

The only citizens that can build barricades are builders, but only during the day. From their position, they can build a barricade in one of the at most 4 adjacent cells (either horizontally or vertically). If a builder is inside a barricade, he cannot build another barricade. A barricade may be occupied by any citizen of the clan who created it, but not by members of other clans. Barricades have an initial resistance, which can be increased up to a certain maximum value by repeated construction operations. There is a maximum number of barricades every clan can own. When night finishes and a new day begins, all surviving barricades disappear.

Warriors. Warriors always carry a weapon. Initially they all carry a hammer, but can take a gun or bazooka. During the day, a warrior can move throughout the board according to the following rules:

- He can only access adjacent cells horizontally or vertically, never diagonally.

- If he moves to a cell occupied by a building, another citizen or a rival barricade, the movement will be ignored.
- If he moves to a cell occupied by a barricade of his clan, the warrior will enter the barricade without destroying it.
- If he moves to a cell occupied by money, his clan will receive the corresponding number of points. The money will disappear and the warrior will occupy the cell.
- If he moves to a cell occupied by food, his life will be increased. The food will disappear and the warrior will occupy the cell. The life of a citizen will never be larger than his initial life.
- If he moves to a cell occupied by a weapon, the warrior will occupy the cell and the weapon will disappear. After that, the warrior will carry the strongest weapon between the weapon in the cell and the weapon he had before the movement. Note that a bazooka is stronger than a gun, and a gun is stronger than a hammer.

At night, the behavior of a warrior does not change, except when he moves to a cell occupied by a rival citizen or by a rival barricade:

- If he moves to a cell occupied by a citizen of the same clan, nothing changes and the movement will be ignored.
- If he moves to a cell occupied by a citizen of another clan, a fight will start. The citizen that is defeated will lose a fixed number of life points. The winner of the fight is determined by the strength of the opponents. Builders always have the same strength, whereas the strength of a warrior depends on the weapon they carry. In a fight between two citizens with strengths N and M, the former will win the fight with probability $N/(N+M)$, whereas the latter will win it with probability $M/(N+M)$. After the fight, opponents will not move but one of them can die if he loses all life points. In this case, the survivor will receive a certain number of points.
- If he moves to a cell occupied by a rival barricade, the barricade will lose as many resistance points as determined by the demolition strength of the weapon he carries. If the barricade loses all resistance points, it will disappear, but the warrior will not move.

Builders. Builders never carry any weapon. At daytime, their movements are the same as the ones of the warriors. The only difference is that when a builder moves to a cell occupied by a weapon, he occupies the cell and the weapon disappears.

At night, the behavior of a builder is the same, except when:

- He moves to a cell occupied by a rival citizen. In this case, a fight starts as explained previously.
- He moves to a rival barricade. In this case, the situation is the same as for warriors, but considering that the demolition strength of builders is very limited.

In general, when a citizen demolishes a barricade where a rival is hiding, the rival does not die, but becomes unprotected and can be attacked. If a citizen is inside a barricade, nobody can attack him. However, he can attack other citizens. During this attack he will have no special protection and hence, could die.

Object regeneration. Whenever food or money disappear, they are regenerated after a certain number of rounds. Weapons and citizens are also regenerated. A weapon is replaced by a weapon of the same type. The same happens with citizens, that are regenerated with the corresponding initial life. Warriors reappear carrying a hammer. In all these cases, the object will appear in an empty cell C such that there is no citizen in the surrounding positions (the ones marked with x in the table).

x	x	x	x	x
x	x	x	x	x
x	x	C	x	x
x	x	x	x	x
x	x	x	x	x

If at the moment of regenerating the object there is no safe cell in this sense, the object will try to reappear in the following round. At each round, citizens will have priority to be regenerated. Then, food and money have the same priority and finally, weapons have the least priority. Hence, if in a round a citizen and a unit of money have to be regenerated but only one safe cell exists, the citizen will reappear and the money will wait until the next round.

Order execution.

At each round one can given more than one order to the same citizen, but only the first one will be considered. Any program that tries to give more than 1000 orders during the same round will be aborted.

At each round, the selected orders of the four players will be executed in a random order, but respecting the relative order among the orders given to the same clan. As a consequence of this fact, consider giving the orders to your citizens from most urgent to least urgent.

Take into account that every order is applied to the board that results from previous orders. For example, consider the board

x	x	x	x
x	M	C	x
x	G	x	x
x	x	x	x

where M represents food, C a builder and G a rival warrior. Imagine that the player that controls C has decide to move it to the left, and the player controlling G moves it upwards. If the order to G is executed first, then C will not eat the money because G has consumed it first. Moreover, the execution of the order to C is an attack to G.

2 The Viewer

In Figure 1 we show a screenshot where all elements of the game appear.

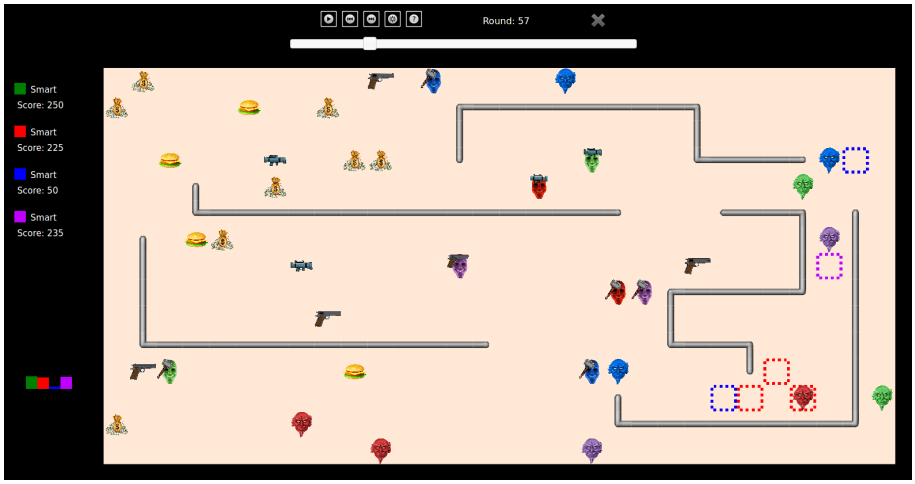

Figure 1: Screenshot of the game.

- On the top left corner there are buttons that allow one to play/pause the match, go to the beginning or the end of the match, toggle on or off the animation mode, or get a help window with further ways of controlling how the match is played. On the top right corner there is a button for closing the viewer. The current round is also shown on the top of the viewer. A horizontal slide bar visually shows in which point of the match this round is.
- In the column on the left, every player appears with the corresponding name and color. In the matches played in Jutge.org, it is also shown the percentage of CPU time that has been consumed so far (if exhausted, this

is indicated with an 'out'). In the lower part of the left column there is a histogram showing the points of every player.

- Street cells are painted light brown at day, dark brown at night. Building cells are marked with grey bars.
- Weapons and bonuses are represented as in Figure 2.
- Citizens are represented as in Figure 3.
- Barricades are represented as in Figure 4.

Figure 2: Representation of a bazooka, gun, food and money.

Figure 3: Representation of a builder and three warriors with hammer, gun and bazooka, respectively.

3 Programming the game

The first thing you should do is downloading the source code. It includes a C++ program that runs the games and an HTML viewer to watch them in a reasonable animated format. Also, a "Null" player and a "Demo" player are provided to make it easier to start coding your own player.

Figure 4: Representation of barricades, from least resistant (left) to most resistant (right).

3.1 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 only need a recent `g++` version, `make` installed in your system, plus a modern browser like Firefox or Chrome.

1. Open a console and `cd` to the directory where you extracted the source code.
2. If, for example, you are using a 64-bit Linux version, run:

   ```
   cp AIDummy.o.Linux64 AIDummy.o
   ```

   If you use any other architecture, choose the right objects you will find in the directory.
3. Run

   ```
   make all
   ```

   to build the game and all the players. Note that `Makefile` identifies as a player any file matching `AI*.cc`.
4. This creates an executable file called `Game`. This executable allows you to run a game using a command like:

   ```
   ./Game Demo Demo Demo Demo -s 30 < default.cnf > default.res
   ```

   This starts a match, with random seed 30, of four instances of the player Demo, in the board designed in `default.cnf`. The output of this match is redirected to `default.res`.
5. To watch a game, open the viewer file `Viewer/viewer.html` with your browser and load the file `default.res`.

Use

```
./Game --help
```

to see the list of parameters that you can use. Particularly useful is

```
./Game --list
```

to show all the recognized player names.

If needed, remember that you can run

```
make clean
```

to delete the executable and object files and start over the build.

3.2 Configuration file

We provide you with two configuration files. In all matches that will be played in Jutge, including the qualifying round and the final, we will use `default.cnf`. This file fixes the parameters of Figure 5 to the default value. The position of buildings, citizens, weapons and bonuses are randomly determined.

Configuration file `default-fixed.cnf` shows how to change parameters and define the board. The characters in the grid are '.'(street), 'B' (building), 'G' (gun), 'Z' (bazooka), 'M' (money), 'F' (food). Regarding citizens, their type is 'w' (warrior) or 'b' (builder), and the possible weapons are 'n' (none), 'h' (hammer), 'g' (gun) and 'b' (bazooka).

You can create your own configuration files for fun or for debugging your players on smaller boards. The parameters have to be within the range defined in Figure 5. Make sure that attack and demolition strengths of a bazooka have to be larger or equal to the ones of the gun, and these larger or equal to the ones of the hammer. Another constraint is that BARRICADE_RESISTANCE_STEP should be smaller or equal to BARRICADE_MAX_RESISTANCE. Finally, if you want to modify parameters and randomly generate the board, make sure that the board is large enough to fit all objects.

3.3 Adding your player

To create a new player with, say, name `Gimli`, copy `AINull.cc` (an empty player that is provided as a template) to a new file `AIGimli.cc`. Then, edit the new file and change the

#define PLAYER_NAME Null

line to

#define PLAYER_NAME Gimli

The name that you choose for your player must be unique, non-offensive and at most 12 characters long. This name will be shown in the website and during the matches.

Parameter	Default value	Range
NUM_PLAYERS	4	$[4, 4]$
NUM_DAYS	5	$[1, \infty)$
NUM_ROUNDS_PER_DAY	50	$[2, \infty)$ (even)
BOARD_ROWS	15	$[12, 25]$
BOARD_COLS	30	$[12, 50]$
NUM_INI_BUILDERS	4	$[1, 6]$
NUM_INI_WARRIORS	2	$[1, 4]$
NUM_INI_MONEY	10	$[0, 10]$
NUM_INI_FOOD	5	$[0, 10]$
NUM_INI_GUNS	4	$[0, 5]$
NUM_INI_BAZOOKAS	2	$[0, 4]$
BUILDER_INI_LIFE	60	$[1, \infty)$
WARRIOR_INI_LIFE	100	$[1, \infty)$
MONEY_POINTS	5	$[1, \infty)$
KILL_BUILDER_POINTS	100	$[1, \infty)$
KILL_WARRIOR_POINTS	250	$[1, \infty)$
FOOD_INCR_LIFE	20	$[1, \infty)$
LIFE_LOST_IN_ATTACK	20	$[1, \infty)$
BUILDER_STRENGTH_ATTACK	1	$[1, \infty)$
HAMMER_STRENGTH_ATTACK	10	$[1, \infty)$
GUN_STRENGTH_ATTACK	100	$[1, \infty)$
BAZOOKA_STRENGTH_ATTACK	1000	$[1, \infty)$
BUILDER_STRENGTH_DEMOLISH	3	$[1, \infty)$
HAMMER_STRENGTH_DEMOLISH	10	$[1, \infty)$
GUN_STRENGTH_DEMOLISH	10	$[1, \infty)$
BAZOOKA_STRENGTH_DEMOLISH	30	$[1, \infty)$
NUM_ROUNDS_REGEN_BUILDER	50	$[1, \infty)$
NUM_ROUNDS_REGEN_WARRIOR	50	$[1, \infty)$
NUM_ROUNDS_REGEN_FOOD	10	$[1, \infty)$
NUM_ROUNDS_REGEN_MONEY	5	$[1, \infty)$
NUM_ROUNDS_REGEN_WEAPON	40	$[1, \infty)$
BARRICADE_RESISTANCE_STEP	40	$[1, \infty)$
BARRICADE_MAX_RESISTANCE	320	$[1, \infty)$
MAX_NUM_BARRICADES	3	$[1, \infty)$

Figure 5: Game parameters. Their explanation can be found in file `Settings.hh`.

Afterwards, you can start implementing the virtual method *play*(), inherited from the base class *Player*. This method, which will be called every round, must decide the orders to give to your units.

You can define auxiliary type definitions, variables and methods inside your player class, but the entry point of your code will always be the *play*() method.

From your player class you can also call other functions that you will find specified in the following files:

- `State.hh`: access the state of the game.
- `Action.hh`: give orders to your citizens.
- `Structs.hh`: useful data structures.
- `Settings.hh`: access the parameters of the game.
- `Player.hh`: method *me*().
- `Random.hh`: generation of random numbers.

You can also examine the code of player "Demo" in `AIDemo.cc` as an example of how to use these functions.

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

3.4 Playing against the Dummy player

In order to try your strategy against the Dummy player, we provide you with its object file. Hence, you do not have access to its source code but can add it as a player and compete against it.

In order to add the Dummy player to the list of registered users, you must edit the `Makefile` and set the variable `DUMMY_OBJ` to the right value. Remind that object files contain binary instructions for a concrete architecture. This is why we cannot provide a single file.

Pro tip: ask you friend their **object** files (never source code!!!) and add them to your `Makefile`.

3.5 Restrictions when submitting your player

When you think that your player is strong enough to enter the competition, you can submit it to the Jutge. 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 (like AIGimli.cc).

- You cannot use global variables (instead, use attributes in your class).
- You are only allowed to use standard libraries like `iostream`, `vector`, `map`, `set`, `queue`, `algorithm`, `cmath`, ... In many cases, you don't even need to include the corresponding library.
- You cannot open files nor do any other system calls (threads, forks, ...).
- Your CPU time and memory usage will be limited, while they are not in your local environment when executing with `./Game`. The timelimit is one second for the whole match. If you exceed this limit (or you execution aborts), your player will be frozen and will not admit any more commands.
- Your program should not write to **cout** nor read from **cin**. You can write debug information to **cerr**, but remember that doing so in the code that you upload can waste part of your limited CPU time.
- Any submission to the Jutge must be an honest attempt to play the game. Any try to cheat in any way will be severely penalized.
- Once you have submitted a player to Jutge that has defeated the Dummy player, you can send more submissions but you will have to change the player name. That is, once a player has defeated Dummy, his name is blocked and cannot be reused.

4 Tips

- **DO NOT GIVE OR ASK YOUR CODE TO/FROM ANYBODY.** Not even an old version. Not even to your best friend. Not even from students of previous years. We will use plagiarism detectors to compare pairwise all submissions. However, you can share the compiled `.o` files.

  Any detected plagiarism will result in an **overall grade of 0** in the course (not only in the Game) of all involved students. Additional disciplinary measures might also be taken. If student A and B are involved, measures will be applied to both of them, independently of who created the original code. No exceptions will be made under any circumstances.
- Before competing with your classmates, focus on qualifying and defeating the "Dummy" player.
- Read only the headers of the classes in the provided source code. 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.
- Define basic auxiliary methods, and make sure they work properly.

- Try to keep your code clean. Then it will be easier to change it and to add new strategies.
- As usual, 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`s to output debug information and add `assert`s to make sure the code is doing what it should do. Remember to remove (or comment out) the `cerr`s before uploading your code to Jutge.org, because they make the execution slower.
- When debugging a player, remove the `cerr`s you may have in the other players' code, to make sure you only see the messages you want.
- By using commands like `grep` in Linux you can filter the output that `Game` produces.
- Switch on the `DEBUG` option in the Makefile, which will allow you to get useful backtraces when your program crashes. There is also a `PROFILE` option you can use for code optimization.
- If using `cerr` is not enough to debug your code, learn how to use `valgrind`, `gdb`, `ddd` or any other debugging tool. They are quite useful!
- You can analyze the files that the program `Game` produces as output, which describe how the board evolves after each round.
- Keep a copy of the old versions of your player. When a new version is ready, make it fight against the previous ones to measure the improvement.
- 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 your competitors by watching matches. This way you can try to defend against them or even improve them in your own player.
- Do not wait till the last minute to submit your player. When there are lots of submissions at the same time, it will take longer for the server to run the matches, and it might be too late!
- You can submit new versions of your program at any time.
- And again: Keep your code simple, build often, test often. Or you will regret.