Battle Royale

Albert Vaca Omer Giménez Jordi Petit Salvador Roura

January 27, 2026

1 Game rules

In this game, each player has control over an army of knights and farmers in a tile-based board.

The goal of the game is to "farm" as many tiles as you can, converting them to your team color. Your score at a given time is the number of tiles of your color at that point, and the winner is the player who has more tiles of his color at the end of the game.

At the beginning of the game, all the knights and farmers of a player are randomly placed at the spawn point of that player: one of the four quadrants of the board.

Each round the players can move each unit one position in one of the 4 cardinal directions.

Moving a farmer to an empty tile will convert the tile to the farmer's team color, even if it already has the color of a different team. Moving a farmer to an non-empty tile is an invalid move.

Moving a knight to a tile occupied by another player's unit will make it attack that unit, performing a random amount of damage. If the health of a unit drops to less or equal than zero, that unit will be converted to the attacking player's team and will respawn at his spawn quadrant with full health. (When this happens, the players can say out loud: Wololo!)

Deliberately not moving a unit will increase its health by a given constant. A unit cannot recover more health than its initial amount. Performing an invalid move will result in that unit not moving, but will not regenerate health.

The boards will have "walls", obstacles that units cannot go through. Trying to move a unit into a wall is an invalid move.

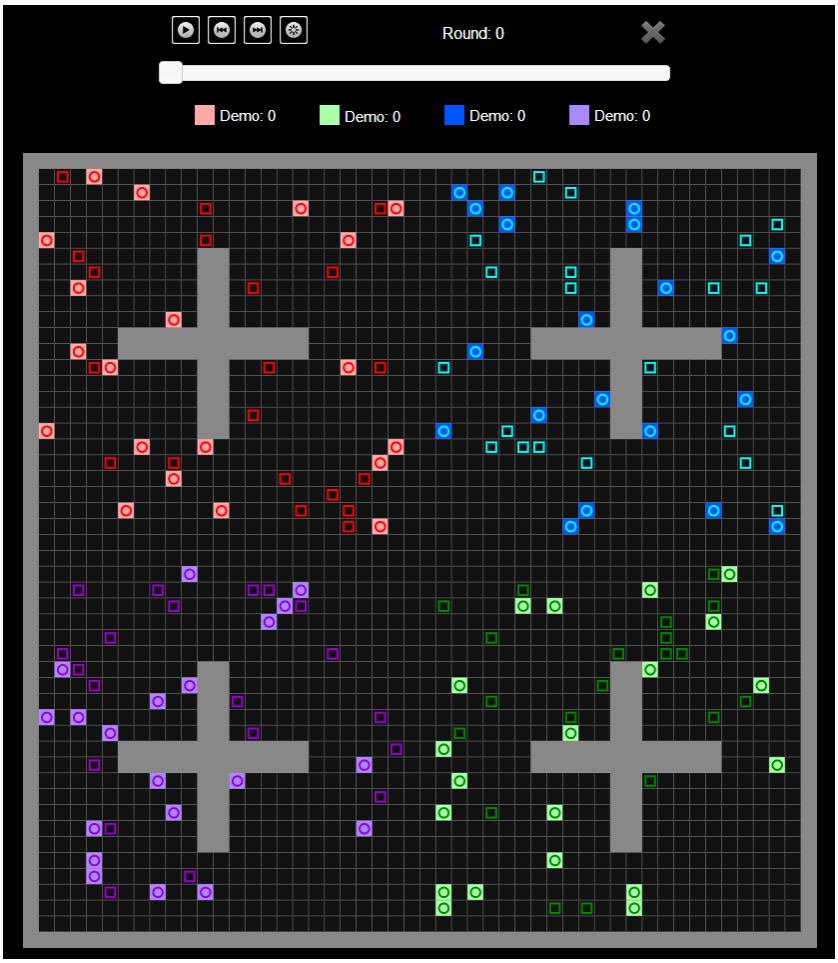
Other invalid moves are moving a farmer to a tile occupied by another unit, or moving a knight to a tile occupied by a unit of the same team.

If the board does not have walls all around, moving out of the board will make a unit wrap to the opposite side.

The order of the actions performed by players is chosen at random, so you cannot expect any execution order of your actions. A notable case is when two units try to move to the same tile: the unit that happens to move first will be able to occupy the tile, while the second one will not.

A game is defined by a board and the following set of parameters:

- nb_players: Number of teams in the game.
- nb_rounds: Number of rounds that will be played.
- nb_farmers: Number of farmers per player.
- nb_knights: Number of knights per player.
- farmers_health: The maximum (and initial) health of a farmer.
- knights_health: The maximum (and initial) health of a knight.
- farmers_regen: The amount of health a farmer will regenerate when not moving.
- knights_regen: The amount of health a knight will regenerate when not moving.
- damage_min: The minimum amount of damage a knight will inflict when attacking.
- damage_max: The maximum amount of damage a knight will inflict when attacking.
- rows: Vertical size of the board.
- cols: Horizontal size of the board.



2 Programming

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

The source code includes a C++ program that runs the games and an HTML 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 our 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 < maze.cnf > game.br
   ```

   Here, we are starting a match with 4 instances of the player "Demo" (included with the source code), in the board defined in "`maze.cnf`". The output of this match will be redirected to "`game.br`".

4. To watch the game, open the viewer (`viewer.html`) with your browser and load the "`game.br`" file.

Use `./Game --help` to see a list of parameters you can use. Particularly useful is `./Game --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

```
#define PLAYER_NAME Demo
```

line to your own player name. 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 inheritance, but do not tell your Software Engineering teachers because they might not like it. The documentation about the available functions can be found both in the header files of the above mentioned classes (and also `PosDir.hh`), and as a PDF you can download from the game website.

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

2.3 Restrictions when submitting your player

When you think your player is strong enough to enter the competition, you should 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 (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, while they are not in your local environment.
- 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 or the implementation.
- Start with simple strategies, easy to code and debug, since this is exactly what you will need at the begining.
- Define simple (but useful) auxiliar methods, and make sure they work properly.
- Before competing with your classmates, focus on defeating the "Dummy" player. This already gives you one extra point!
- 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 `assert`s to make sure the code is doing what it should do. Remember to remove the `cerr`s 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 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 JPlag and other 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!
- And again: Keep your code simple, build often, test often. Or you will regret.