Minesweeper with a Twist!

Nicolas Pigadas
Department of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
el18445

***Abstract*—Minesweeper is a classical computer game that has been popular since its inception and creation in the '60s. The objective of the game is to uncover every tile of a rectangular grid containing hidden mines, without detonating any of them. The game is played by revealing the contents of those individual tiles in the grid, which can contain either a mine or a number representing the number of adjacent mines. Minesweeper has become a beloved classic due to its simple yet challenging gameplay, as well as its ability to offer a wealth of strategic depth, requiring players to think carefully about each move and make decisions based on limited information. This paper will explore the implementation of the classical game, as well as the addition of extra features and enhancing techniques, that can make the game more captivating and intriguing.**

***Index Terms*—Minesweeper, Java, JavaFX, Game Enhancement, Classical Games**

I. The Game

A. Game's Logic

Let's first explore the original game's logic:

- Minesweeper is played on a rectangular grid of squares, some of which contain hidden mines.
- Your objective is to clear all the squares without detonating a mine.
- Click on a square to reveal whether it contains a mine or not. If it doesn't, it will reveal the number of neighboring squares that do. If the number is 0, many squares will be revealed recursively.
- Use the numbers to deduce where the mines are located.
- Optionally, right-click on a square to mark it as a potential mine.
- If you reveal a mine, the game is over and you lose.
- If you clear all non-mine squares, you win the game.
- Note: The game can be customized by adjusting the size of the grid and the number of mines.

B. Game's Twist

Let's examine the game's 'twisted' logic:

- You can add the mode of a super bomb, a bomb that when flagged within the first four tries, its row and column are revealed.
- Using the quick play menu items you can instantly start a new game, just by selecting one of the options: Size, Levels, Time. A perk of those options is that they can't be defined in a user scenario and that difficulty (levels) is defined by the percentage of bombs placed in the given grid.
- Apart from the quick play menu items, you can also customize the game yourself:
  1) Use 'Create' option in 'App'
     a) Add scenario's ID
     b) Add level difficulty: 1 for easy, 2 for difficult
     c) Choose the number of sea mines: between 9-11 for difficulty 1 and 35-45 for difficulty 2
     d) Choose the available time provided: between 120-180 for difficulty 1 and 240-360 for difficulty 2
     e) Select the existence of a superbomb: 0 for difficulty 1 (non-existent) and 1 for difficulty 2 (existent)
  2) Use 'Load' option in 'App'
     a) Add scenario's ID 3) Use 'Start' option in 'App' and enjoy your game. Also, look for future updates; we will make the game even more customizable.

Note: Expanding the storyline of the submarine, the sea mines and the enemy/friendly submarines is in the works of being integrated in the gameplay and further expanding and ameliorating the game's plot and functions.

II. The Game's Scenario

I will let the game's description speak for itself:
"In this game, you are in a submarine in the enemy's territory! Welp! The sea mine radar doesn't work anymore... Thankfully, we are equipped with a radar that shows the sea mines around each square (kilometer). You have to locate every sea mine and flag it in order to help our submarine devise its course and maneuver to safety! But beware... The enemy submarines are onto you! They are on your tail and about to catch you, be fast!"

III. The Game's Implementation

The game was implemented using Java and JavaFX libraries on Intellij.

A. Game's Design

The game's aesthetics are designed in such a way that -in combination with the game's description- give the player the feel that the story takes place underwater. The tiles are blue, the bombs are sea mines. The game over, surrender, timeout and winning pictures are correlated with the ongoing theme of the submarine escape. The sounds were picked to match those aesthetics.

B. Game's Behavioral Design: Behind the Scenes

The game's flow is designed in such a way that magnetizes the users, keeping them in a rush. Firstly, the game starts immediately when the application starts running. When you lose on time, the auxiliary timeout pop-up doesn't stop the countdown, so the sense of urgency is kept high at all times, but when you lose in another way, the flow of the game stops, because we don't want to amplify the player's frustration. To be more specific, losing on time starts a game immediately, without having the chance to see the solution or catch your breath, renewing the challenge in the player's mind and provoking them to continue, in the midst of their adrenaline. Yet, there is a pause button, so that there is a feeling of security of not losing the game progress. Finally, the game can be restarted with different options (Size/Levels/Time) instantly, just by using one of them in the menu item and then the game instantly restarts. In that way, that the gaming experience flow isn't interrupted. An integrated perk is that the options given in the menu items aren't allowed in the user defined scenarios!

IV. Project's Object-Oriented Structure

For the OOP design I kept in mind two rules:
1) Group related fields and methods in the same object-entity.
2) Don't over-complicate the structure. For example, the three types of game overs (sea mine, timeout, surrender-solution request) could have been grouped together to form the 'GameOver' and the winning method to be the 'Win' class. Yet, the complexity of the Main-Application class required to group inside it the logic of the flow of the game, so each of the pre-mentioned methods stayed as such, within the class Application, along with the rest of game flow altering and refresh methods (pause, unpause, reload, createContent).

A. File, Class & Method OOP Structure

The file structure represents the high level organization of my OOP design choices and the class and method organization reflect the low level design I had in mind:

- **Main.java**: Contains the application's most important functions.
  class Main extends Application: Includes the methods for the functionality of the application:
  1) method start(): creates the application's window and every menuItem's functionalities. Calls reload() and createContent().
  2) method reload(): Resets every variable related to a game and sets the ones for the next. Then, reload() calls createContent().
  3) method createContent(): Creates grid, sets bombs and tiles' numbers.
  4) method gameOver(): Game over due to left click on sea-mine. Informative pop ups informs the player.
  5) method TimeLimit(): Game over due to time-out. Called when countdown reaches zero. Doesn't show solution to hook the player and keep him playing (see section III.B. Game's Behavioral Design). Informative pop ups informs the player.
  6) method gameOver3(): Game over due to surrender. The player requested the solution. Informative pop ups informs the player.
  7) method win(): The player wins! Informative pop ups informs the player.
  8) method pause(): Freezes tiles and stops countdown until unpause is pressed.
  9) method unpause(): Unfreezes the tiles and resumes the countdown.
  10) method main(): Launches the application/game.

- **scenarioReader.java**: Reads a user-defined game's scenario from txt file and handles exceptions accordingly.
  class InvalidDescriptionException: Handles scenario description error:
  1) method InvalidDescriptionException(): The method handling scenario description error (file doesn't contain exactly 4 lines) and prints a message to the console.
  class InvalidValueException: Handles scenario value error:
  1) method InvalidValueException(): The method handling scenario value error (one of the four values isn't within the permitted range) and prints a message to the console.
  class scenarioReader: Reads scenario and handles io, file reading, value and description errors in the scenario:
  1) method readScenarioFile(): The method handling scenario value error (one of the four values isn't within the permitted range) and prints a message to the console.
- **scenarioWriter.java**: Writes a user-defined game's scenario to txt file and handles exceptions accordingly.
  1) method writeScenarioFile(): Implements is classes purpose.
- **Tile.java**: Every tile is a button that can uncover various graphics, following the game's logic.
  class Tile extends StackPane: Represents a Tile in a game of Minesweeper and all the actions it might happen to it, as well as every design it might change to.
  1) constructor Tile(): The Tile constructor constructs a Tile using its x and y coordinates, as well as the information for the existence or not of a bomb behind the Tile.
  2) method onClick(): The method that defines what will happen to the Tile when it is clicked.
  3) method blankClick(): The method that recursively reveals adjacent tiles of a blank tile.

B. The Rest of the Files

src: Contains the java and css files, the image and sound multimedia used in the game and the javadocs folder with its js, html, css and multimedia files.

gameplay images: Contains images of the gameplay.

javadoc_command.txt: Contains the javadoc commands to create html indexes for the javadoc browser documentation.

Minesweeper-master.iml: The Minesweeper-master.iml file is an IntelliJ IDEA module file. It is generated by IntelliJ IDEA when you create a new project or module, and it stores information about the project's modules, libraries, and dependencies.

current game mines: Contains **mine.txt** which contains the bomb and super bomb positions in the current game.

scenarios: Contains the user-created and some pre-loaded customized game scenarios .

out: When you run the Minesweeper JavaFX application in IntelliJ IDEA, the compiled class files and any other files generated during the build process will be stored in the out folder. One of the main files generated by the Java process is the bytecode file, which has the extension ".class" and contains the compiled code of the Java classes. It also includes any resource files (such as images or configuration files) that are part of your project and may generate log files and other output files, depending on the configuration of your application and any third-party libraries or frameworks that you are using.

V. Documentation

The projects is documented in three ways:

- The code accompanied by explanatory comments.
- Javadocs document the project's fields, methods and classes.
- The present paper, which presents an overview of the project's logic and development.

VI. Conclusions

In summary, in this work I implemented Minesweeper using Java and JavaFX following Object-Oriented Programming principles and added extra features and capabilities. I also imagined a scenario about the theme of the game revolving around a submarine that has to locate the sea-mines in order to escape from the enemy's territory and submarines. Finally, I adapted the aesthetics to match the scenario and documented fully every aspect of the project project.

VII. Future Work

The future work of this work seems pretty intriguing and exciting. While developing it, many ideas came to mind that would add more things to do in the game and expand the scenario. The first idea is about drawing the course of the submarine from a starting point, trying to avoid the sea mines, as well as the courses of enemy submarines that enter the grid from different tiles. For example the starting point could be on the left side of the grid (0, y) and the finish point on the right one (x, 0). The course might be a line, or a path of sequential tiles. Even if the countdown stops, the player will still be able to devise a course for their submarine, but blindly, so there are many chances of failure but the hope of winning is kept alive!

When this final stage of the game starts, the submarine and enemy and friendly submarines will follow their course on the grid, adding animation aspects to the game, as the player will watch his plan unfold live. The enemy and friendly submarines will fight and both be destroyed if their courses meet, freeing the course of the player's submarine. Or even there can be a difference between the submarines' powers and the player can take up the task to design all the friendly submarines' courses including his own, being unaware of the enemy submarines' positions. Also, instead of just making the grid bigger, we can make it scrollable and really big, just to add the sense of exploring the map while trying to figure out the tiles.

Moreover, in this big scrollable map we can add many features like the super bomb, for example if you left click an anti-mine tile, if you hit a sea mine that is adjacent to it, you are saved. This way, the player will feel the relief of just escaping loss. Or even, if you click 10 neighboring tiles in the same row, all the adjecent tiles are revealed. Also, the scenario can improve the games visuals, for example the blank tiles can be safezones where friendly submarines offer temporal protection and that can be visually implemented.

My general idea is that on the grid there will be multiple items, and in the game finale, the player will see everything animate, leading the submarine to its escape and the player's consequent win or its demise and the game being over.

VIII. Requested Features Implementetion Overview

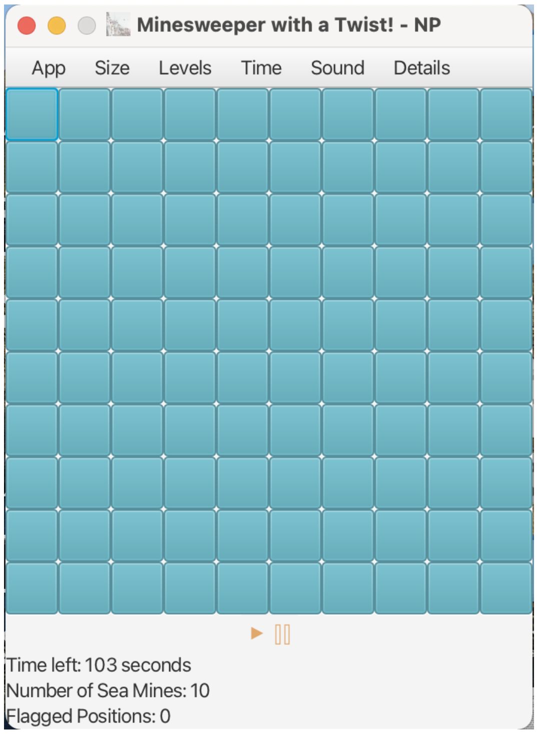
Feature	Nested Features	Implemented?	Image in dir: gameplay images
Classical Features		✓	
	Menu Bar, Grid	✓	1
	Info about current game	✓	1
	Flagging tiles	✓	6f
	Flag no. limit (== no. of bombs)	✓	
	Flag disables tile	✓	
	Random bomb placement	✓	
	Changing bomb placements	✓	
	Number assignment to tile	✓	6f
	Recursive tile reveal	✓	6f
	Bomb gameover w/ popup	✓	6b
	Win w/ pop-up	✓	6f
	Mouse clicks w/ corr. actions	✓	
Extra Requested Features		✓	
	txt scenario writer	✓	
	txt scenario reader	✓	
	Valid scenario constraints	✓	6g
	InvalidDescriptionException	✓	
	InvalidValenException	✓	6g
	Bombs pos. in mine.txt	✓	6c, 6f
Modification Features		✓	
	Countdown Loss	✓	6e
	Super Bomb	✓	6h
User Interface		✓	Im. 1
	Main Window and its dimensions	✓	1
	Current Game: No of total bombs	✓	1
	Current Game: no of placed flags	✓	1
	Current Game: Countdown	✓	1
	Grid and button tiles	✓	1
	Left click: tile reveal, UI update	✓	6f
	Right click: placing flag, UI update	✓	6h
	MenuBar(App, Size, Levels, Time, Sound, Details)	✓	1
	MenuItem App(Create, Load, Start, Exit)	✓	Im. 2a
	MenuItem Details(Rounds, Solution, About, Rules, Information)	✓	Im. 2f
Other requirements		✓	Im. 1
	OOP Design	✓	
	Javadoc Documentation	✓	9
	IDE: Intellij files	✓	9
	Present report	✓	

TABLE I: All the request features are implemented, as shown above.

IX. Extra Features Implementetion Overview

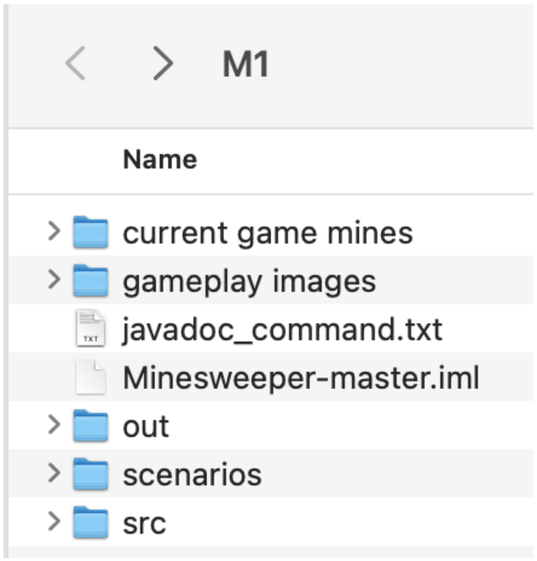
Feature	**Nested Feature**	**Implemented?**	**Image in dir: gameplay images**
Game's Flow		✓	
	Pause/Unpause button	✓	3
	Reloading after a game finishes	✓	
	Quick new game from menuItems Size/Lavels/Time	✓	2 b,c,d
User Interface		✓	
	Images and sounds in game events (win, loss, surrender)	✓	
	MenuItem Size(10x10, 15x15, 20x20)	✓	2b
	MenuItem Levels(Easy, Medium, Hard)	✓	2c
	MenuItem Time(120, 180)	✓	2d
	MenuItem Sound(On, Off)	✓	2e
	MenuItem Details(About): About the game dev	✓	7a
	MenuItem Details(Rules): About the game's rules	✓	7b
	MenuItem Details(Rules): About the game's scenario	✓	7c

X. Screenshots for Visualisation



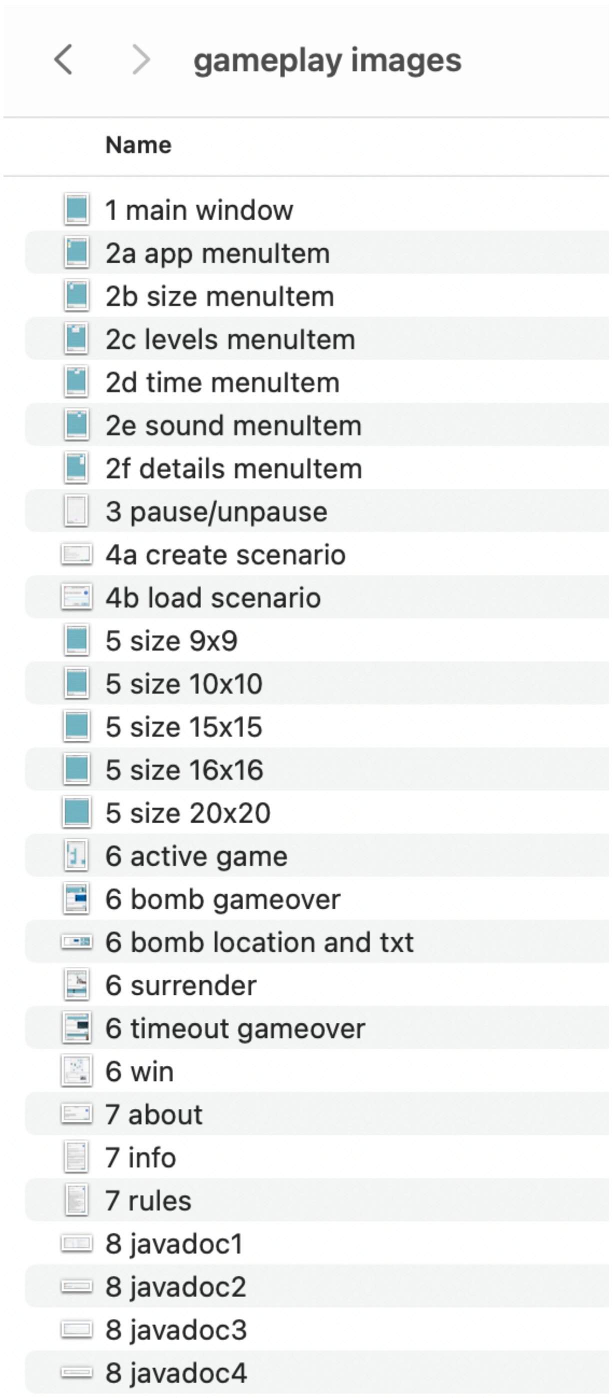
Time left: 103 seconds
Number of Sea Mines: 10
Flagged Positions: 0

Fig. 1: Main window



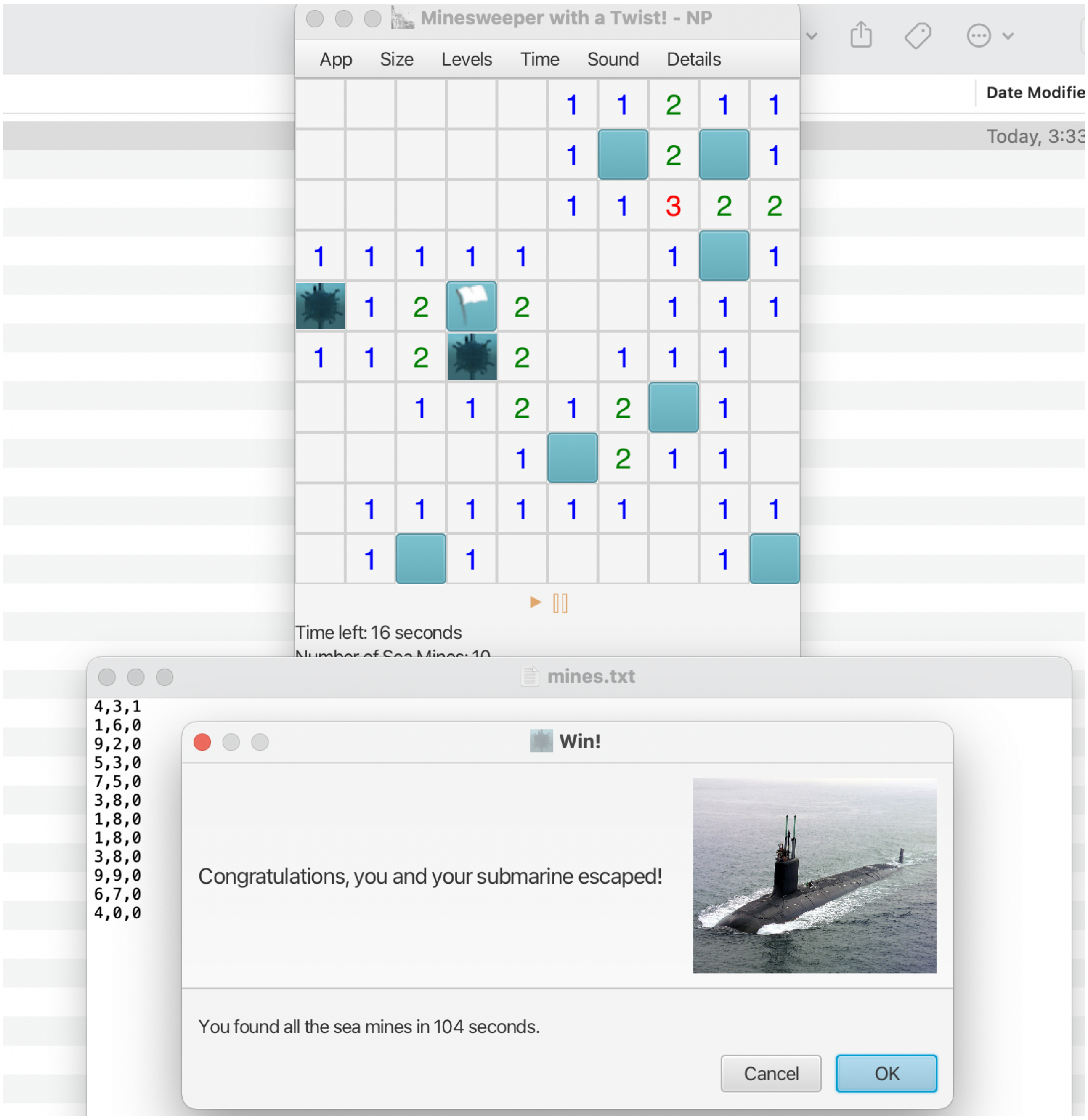
current game mines
gameplay images
javadoc_command.txt
Minesweeper-master.iml
out
scenarios
src

Fig. 2: Files apart from the present report, which was added later.



1 main window
2a app menuItem
2b size menuItem
2c levels menuItem
2d time menuItem
2e sound menuItem
2f details menuItem
3 pause/unpause
4a create scenario
4b load scenario
5 size 9x9
5 size 10x10
5 size 15x15
5 size 16x16
5 size 20x20
6 active game
6 bomb gameover
6 bomb location and txt
6 surrender
6 timeout gameover
6 win
7 about
7 info
7 rules
8 javadoc1
8 javadoc2
8 javadoc3
8 javadoc4

Fig. 3: Gameplay images



Time left: 16 seconds

Number of Sea Mines: 10

```
4,3,1
1,6,0
9,2,0
5,3,0
7,5,0
3,8,0
1,8,0
1,8,0
3,8,0
9,9,0
6,7,0
4,0,0
```

Congratulations, you and your submarine escaped!

You found all the sea mines in 104 seconds.

Fig. 4: Winning