**TEAM “TROLL”**

*Project Documentation*

Table of Contents

1. [**Team Members** 2](#_Toc377648708)
2. [**Project Description** 2](#_Toc377648709)
   1. [*Structure of the project* 3](#_Toc377648710)
   2. [*General rules of the game* 3](#_Toc377648711)
   3. [*Technical documentation* 4](#_Toc377648712)
3. [**TFS Repository** 4](#_Toc377648713)
4. [**Additional Information** 4](#_Toc377648714)



1. **Team Members**

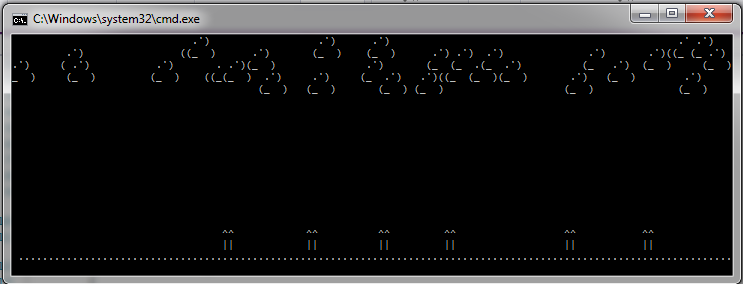
*All six members of team “Troll” are listed in the table below (in alphabetical order).*

|  |  |
| --- | --- |
| *Name* | *Student System Username* |
| Blagoy Shokov | Blagoy.Shokov |
| Biser Sirakov | BiSirakov |
| Zlatimir Mihailov | mihaylov |
| Krasimira-An Petrova | kavpetrova |
| Kristian Ivanov | Kristianc |
| Nikola Bogomirov | loderunner |
| Petko Hadjipenkov | petkoH |
| Simeon Tzvetkov | STzvetkov |

1. **Project Description**

Our team had several different ideas for console games which could meet the requirements of the assignment. We chose to implement the one we called “**Troll runner“** – a game where future (Google Chrome default game when there is no connection) meets the past (the good old Moon Racer).

A draft version of the plain over which our troll is running and jumping was created first which allowed us to visualize how the game would look on the console. You can see how the troll’s environment (both ground and sky) looks like on the picture below.



The strange forms in the upper part of the screen are clouds and some very rare pickups. Bellow, on the ground, we have the obstacles that our troll has to jump over.

To organize our work on the project, we discussed the rules of the game with all their implications which needed to be implemented. We developed the overall structure of the project (its classes, methods and the general logic) and distributed the different tasks between the team members.

* + Structure of the project

The game’s design follows the main concepts of the object-oriented programming. There is a corresponding class for each object from the real world: path, obstacle (land and air), pickup, troll, player, etc. Some basic inheritance is implemented.

The main functionalities which have been implemented are:

* visualizing the “running path” in the console;
* movement (running + jumping) along the path;
* collecting pickups and represent their impact over game behavior;
* loading objects’ drawing design from files;

For more detailed information on the project’s architecture please refer to 2.3: Technical documentation.

* + General rules of the game

After a thorough investigation of the two games that inspired our one (hours and hours spent in “running”, jumping and shooting) we got the main playing concepts. Then we merged them and let our imagination go wild. The final result may cause a mild heart attack to the authors of the original games if they see what we have done with their ideas.

These rules form the overall logic of our program.

1. The game is a single player one. Adding a second “troll” to share the same keyboard could only lead to real life bloodsheds and even worse: unwanted damage of the input device.
2. The player uses **up arrow** to jump and **right arrow** to increase movement speed. **Space** is used for shooting.
3. .
   * Technical documentation (classes and their responsibilities)
     1. class Game
        1. Short description

* Contains all the information for a single game. All main objects are created and used here. Includes the entry point for the program - method Main()
  + - 1. Important methods
         * static void InitializeGame()
* Set default values for:
  + - * + static void Draw()
* Draws the whole playing ground in the console
  + - * + static bool DetectCollision(in List<LandObstacle> traps, in List<Pickup> pickups, out LandObstacle collidedLandObstacle, out Pickup collidedPickup)
  + static void EndGame()
  + static void AcquirePickup()
    - * + static void ProcessCommandsTillEnter()
* Handle user commands by calling the appropriate methods.
  + 1. class Path
       1. Short description
* Represents the running ground.
  + - 1. Important methods
* public static void DrawPath ()
* Draws the actual state of the playing ground in the console
  + 1. class Runner
       1. Short description
* Represents the “troll”. Each player has four of them and moves them on the board when s/he is on turn.
  + - 1. Important methods
* public void Draw(int row, int col)

* + 1. class Obstacle
       1. Short description
* .
  + - 1. Important methods
* + 1. class LandObstacle
       1. Short description
* .
  + - 1. Important methods
* .
  + 1. class AerialObject
    2. class Pickup //to be done
    3. class ExceptionImpossibleMove
* Derives the standard class Exception
* Serves to handle the case when the requested movement of the troll cannot be performed due to abnormal situation (impenetrable aerial obstacle)
  + 1. class ExceptionImpossibleMove
    2. class LoadGraphics
       1. Short description
* Load the graphics about all visualized objects from the corresponding files.
  + - 1. Important methods

1. **GitHub Repository**

*This is the URL of our GitHub repository (****Project “Troll-runner”****):*

[https://github.com/STzvetkov/Troll-runner.git](https://tfs.codeplex.com:443/tfs/TFS36)