**Assignment #2: NumberCruncher**

**Computer Programming Enhanced Concepts**

***Functional Specification***

|  |  |
| --- | --- |
| Student: Frederic Melanson |  |
| Instructor: Stephen Johnson |  |
| Information Technology: Electronic Gaming Development |  |
| May 23, 2017 |  |

# Table of Contents

[Table of Contents 2](#_Toc483315682)

[Requirements Summary 3](#_Toc483315683)

[System Requirements Summary 5](#_Toc483315684)

[Use Case Design 6](#_Toc483315685)

[Class Diagram Design 10](#_Toc483315686)

[Activity Diagram Design 11](#_Toc483315687)

[Assumptions and Dependencies 12](#_Toc483315688)

[Feature Cuts and Unsupported Scenarios 12](#_Toc483315689)

[References 13](#_Toc483315690)

# 

# Requirements Summary

Game:  NumberCruncher

Game Overview:

'NumberCruncher' is a variant of the 'Guess-That-Number' game.  In 'Guess-That-Number', the user is asked to guess a value within a certain range of values.  For instance, the user would be asked to guess a number between 1 and 10.  In most implementations of the game, the user is provided several attempts to guess the number.  If they successfully guess the correct number, they win.

'NumberCruncher' will be like that of 'Guess-That-Number', however, it will have increasing complexity by guessing multiple numbers in the range, and increasing sizes of ranges as the levels increase.

 Mechanics:

· The player will be welcomed to the game (splash screen)

· The player will be asked to provide their initials, for recording high scores

· The player will be asked to select one of three starting levels (easy, moderate, difficult)

o Easy will start with a range of 1..10, three guess 'tracks', and a limit of 5 bad guesses per track

o Moderate will start with a range of 1..100, five guess 'tracks', and a limit of 7 bad guesses per track

o Difficult will start with a range of 1..1000, seven guess 'tracks', and a limit of 11 bad guesses per track

· Based on the starting level, the application will generate a list of 1000 random numbers within the specified range.  The list will then be analyzed to determine the statistical mode of the dataset.  The mode will become the number the player will try to guess.  Note that this process will need to be completed for each of the 'tracks'.  If more than one mode is determined in the dataset, a new dataset is generated.

· The player will guess each 'track' in succession.  The player area will display the value they selected, and then display an 'X' if they guessed the wrong value and a checkmark if they guessed the correct value.

· In addition to the 'X' or checkmark, an arrow will accompany the results to assist the player in determining if the value they provided is too low, or too high.

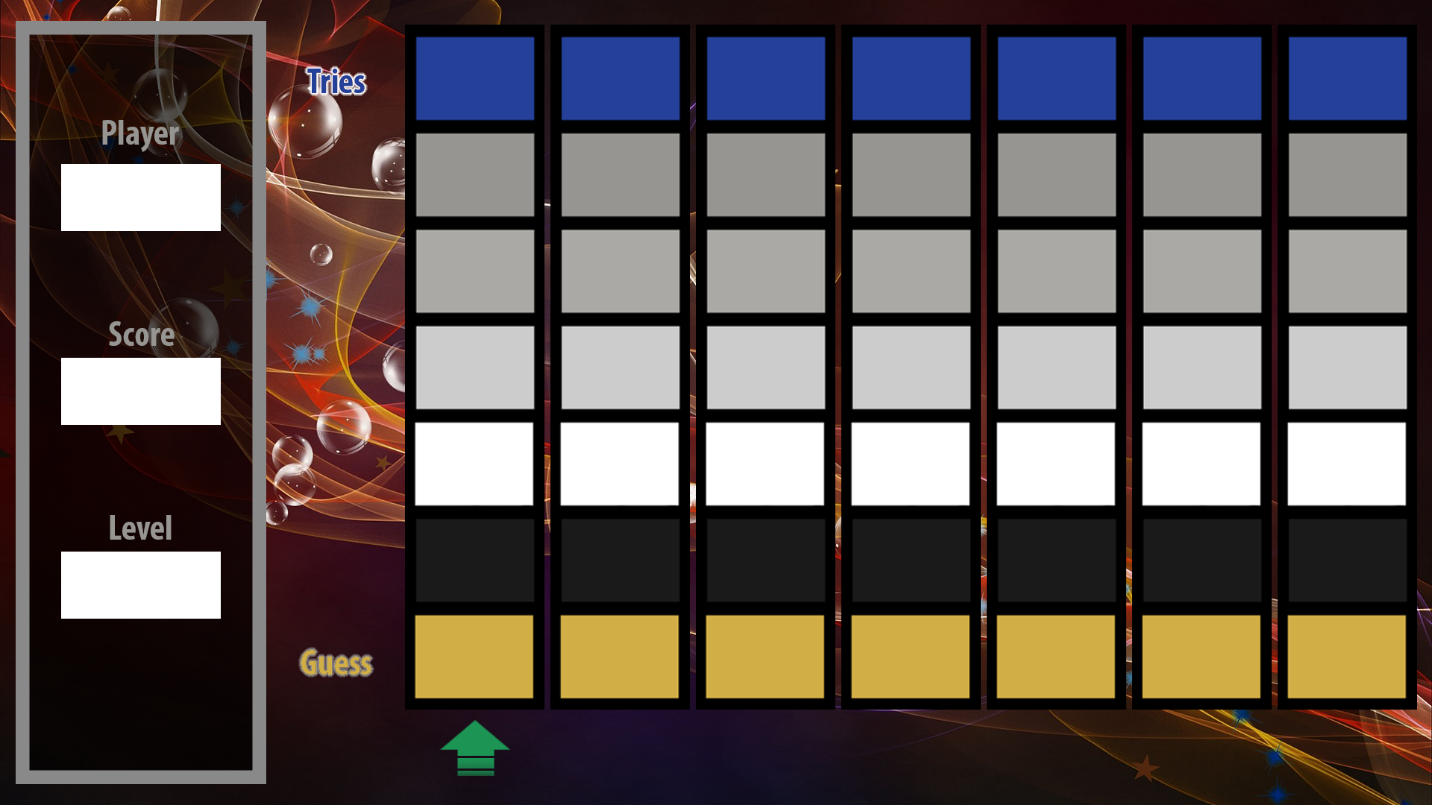
· Guesses that are not used are moved over to the next level.  (Example: if the player guesses the correct number in 2 tries, and was permitted 3 tries, the unused guess is transferred to the next level)

· Each subsequent level's range is increased by 10 (for 10,20,30,40,50,60,70,80,90), 100 (100,200… 900), and so on.

Scoring:                    Scoring will be determined based on the spare guesses that the player has when the level is complete.  Each spare is worth 10 points, however, a bonus of 50 points is provided for every 3 spares that are accumulated in the level.

Game State:             The player's state is maintained by a user profile containing the player's initials, their score, and the levels that they have completed.

User interface:



# System Requirements Summary

**Hardware requirements:**

* **Processor:**

1.5 gigahertz (GHz) or faster processor

* **RAM:**

New device: 2 gigabyte (GB) for 32-bit or 64-bit

* **Hard disk space:**

22 GB

* **Graphics card:**

DirectX 9 or later with WDDM 1.0 driver

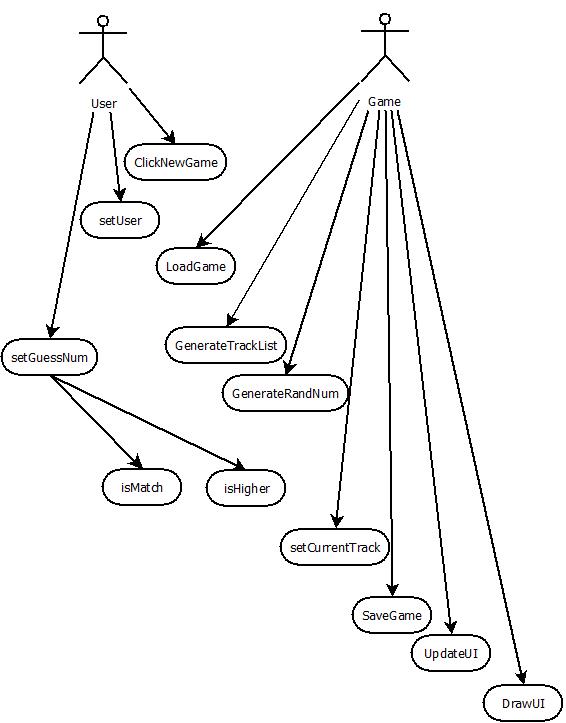
* **Display:**

800x600

**Software requirements:**

* **Oracle Java SE Development Kit 8u131**
* **Eclipse Neon.2 Release (4.6.2)**
* **Latest version of Windows 10 64-bit**
* **DIA diagram tool v0.97.2**
* **Microsoft Word 365**
* **Photoshop CC 2017 1.1 release**

# Use Case Design



**Name:** ClickNewGame

Participating actors: User

Entry conditions:

* Application is running

Exit conditions:

User clicks the “New Game” button

**Name:** setUser

Participating actors: User

Entry conditions:

* New Game has been chosen

Exit conditions:

User enters his id and clicks ok.

**Name:** LoadGame

Partipating actors: Game

Entry condition:

* User has entered his ID

Exit conditions:

If the user ID exists, do a validation (is this you?), else create a new file for new user.

Name: GenerateTrackList

Participating actors: Game

Entry conditions:

* No track list appropriate for the current level exists

Exit conditions:

A TrackList is generated.

Event flow:

* When the game is launched, a dynamic splash screen is displayed with a New Game button.
* When the user clicks the button, he is prompted to enter an identifier. (nickname, initials etc.)

Event flow:

* When prompted, the user will enter his unique identifier.
* If the identifier matches one of those on file, the user will be asked to confirm if it is him.
* If the identifier is not on file, a new file is created.
* If the identifier is on file, but it is not this user’s file, the user will be asked for a new identifier.

Event flow:

* Using the user’s identifier, the game will load a file containing the user’s current level, score, and remaining guesses per track.
* If the user is not on file, a new file is created.

Event flow:

* Depending on the level, the game generates a list of 3, 5 or 7 track objects each with a distinct random number and its own “tries” counter.

Name: GenerateRandNum

Participating actors: Game

Entry conditions:

* Generating a new level

Exit conditions:

A random number is generated.

Name: setGuessNum

Participating actors: User

Entry conditions:

* No match has been made on current track

Exit condition:

User has entered a guess.

Name: isMatch

Participating actors: Game

Entry condition:

* User has entered a guess

Exit Condition:

Returns true if guess matches the random number.

Name: isHigher

Participating actors: Game

Entry condition:

* isMatch returns false

Exit conditions:

Returns true if guess is higher than the random number.

Event flow:

* Generates a list of 1000 random numbers between one and the current maximum value
* Take whichever number appears most often on the list
* If two numbers have the same appearance frequency, start over.

Event flow:

* User inputs his guess in the appropriate track

Event flow:

* Check user input against random number.
* If they match returns true
* If not returns false

Event flow:

* Check user input against random number
* If user input is higher than random number, return true
* If not, return false

Name: SaveGame

Participating actors: Game

Entry conditions:

* User has completed a level

Exit conditions:

Current level, score and remaining tries for every track are saved in a file.

Name: UpdateUI

Participating actors: Game

Entry conditions:

* Information has changed

Exit Conditions:

Changed information is updated and ready to be displayed

Name: DrawUI

Participating actors: Game

Entry conditions:

* UI has been updated

Exit conditions:

Draw the newly updated UI.

Name: setCurrentTrack

Participating actors: Game

Entry conditions:

* Player has entered a guess on a track

Exit conditions:

A new track is set as the current track.

Event flow:

* Saves level, score and remaining tries for each track in the user’s save file.

Event flow:

* Updates all the information in the user interface.

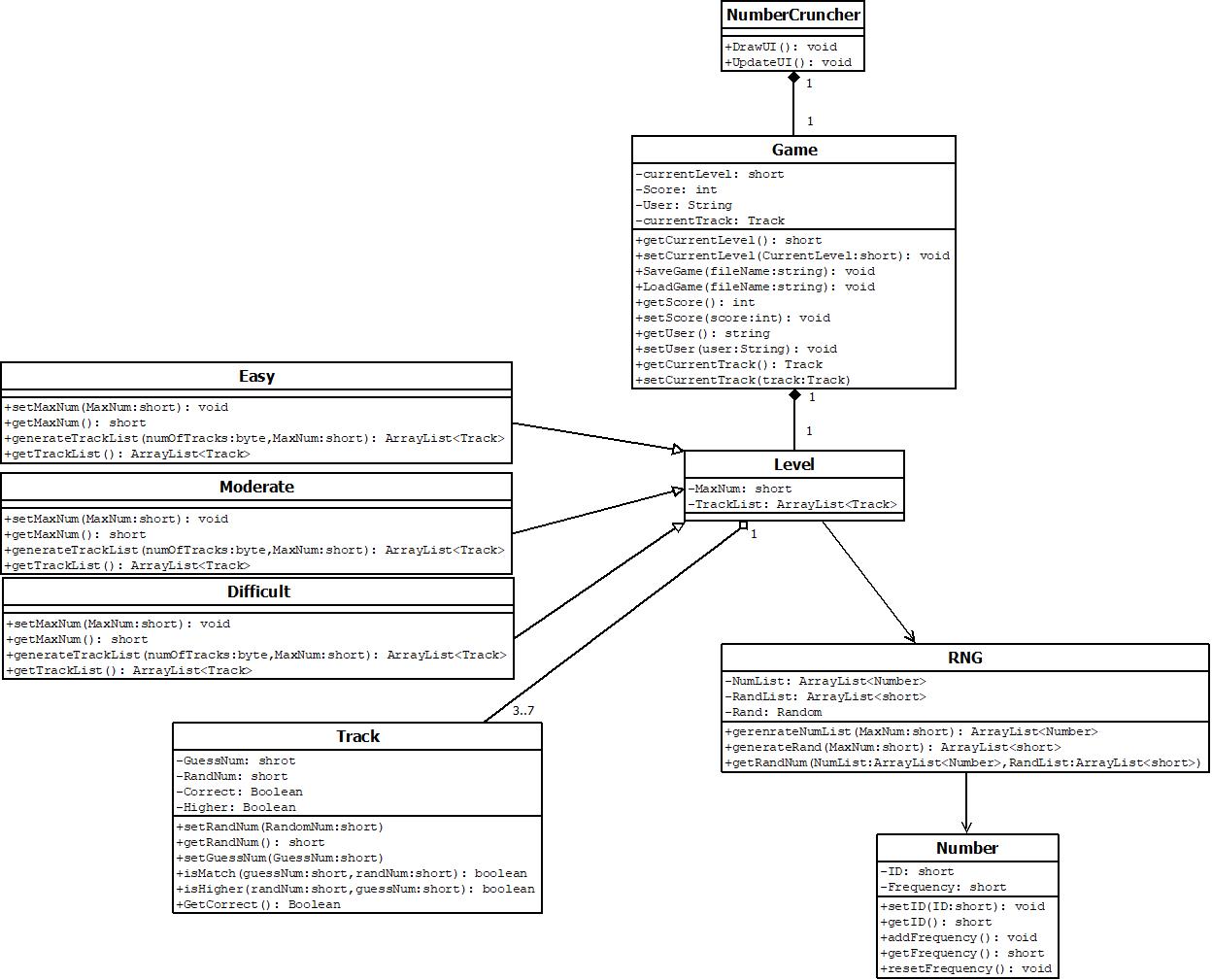
Event flow:

* Displays the user interface.

Event flow:

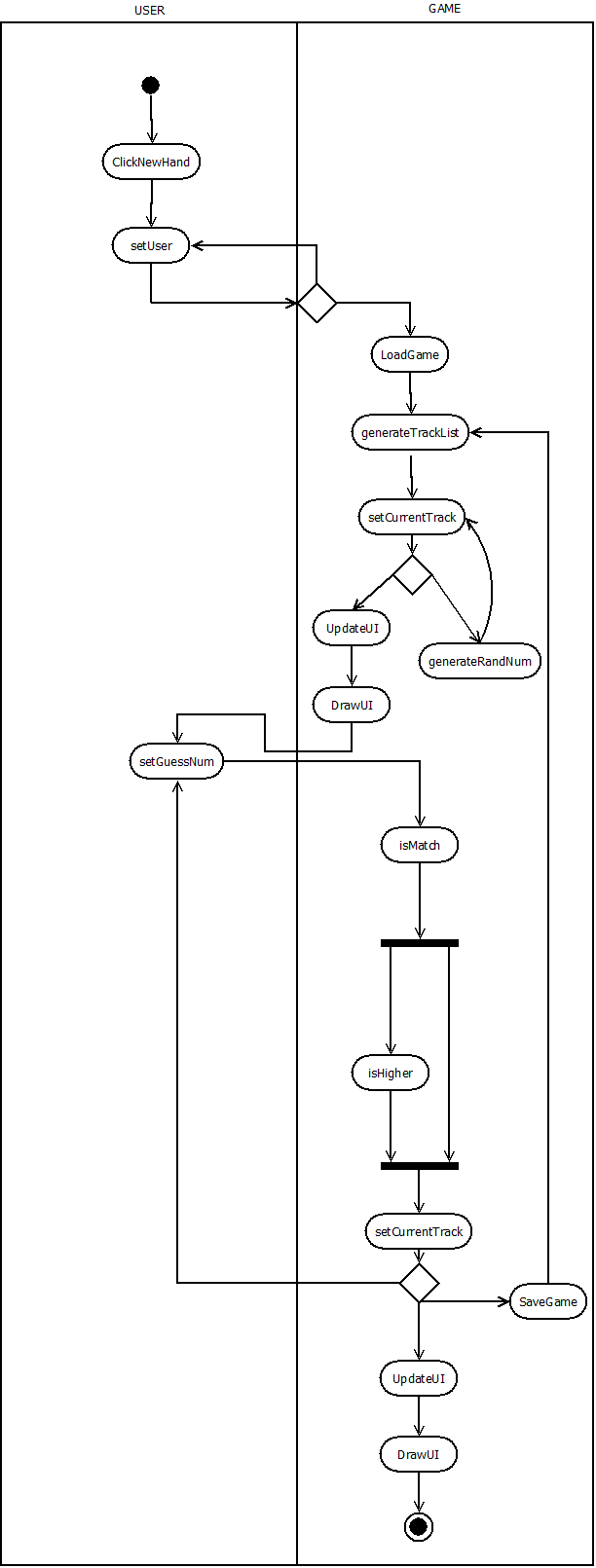
* Once the user has guessed a number on a track, the current track is set to the next one in the track array.
* If the current track is the last in the array, set current track to the first one in the array.
* If all tracks are completed, generate next level

# Class Diagram Design

Number Cruncher will have 6 classes, one of which (the level class) will have three subclasses. The “NumberCruncher” class will handle all the displaying and updating of the user interface, the “Game” class will handle the game mechanics such as scoring and our current level, the “Level” class manages all the guessing tracks while “Track” handles the guessing and matching of numbers. “RNG” generates a random number, and “Number” is used to get our modes in “RNG”.

# Activity Diagram Design

Once the user clicks the new game button and inputs their user id, the last level they started is loaded. A new set of tracks is generated, each with a new random number. The User then inputs a guess for the first track, the game checks if it is a match, if it isn’t, it checks if the guess is higher or lower. The game then switches to the next track and starts over. If all tracks are completed, the game saves and generates the next level.



# 

# Assumptions and Dependencies

The lack of tutorial in this game assumes the player has read the present document in order to know how the game works.

# Feature Cuts and Unsupported Scenarios

Some parts of the prototyped user interface suggested in the “Requirement summary” of this document may not be implemented.

# References

<https://www.microsoft.com/en-ca/windows/windows-10-specifications#sysreqs>

<https://docs.oracle.com/javase/8/docs/technotes/guides/install/windows_system_requirements.html#BABHCGGA>