Project 2 Specification

AwesomeSquare Arcade Game

Your project (game) **must implement the functionality (skills) asked for in this specification**. You should **talk** to your lecturer if your game does not, to make sure it is suitable.

The coding will be done in sections. You will be asked to demo sections of the project in the lab. We are giving you the basic design of the classes. You are to use the class structure as given in the file **Project 2 Class Structure.doc** on your common drive. If you want to use a different class structure for the game, you need to talk to your programming lecture in advance of doing so. This is to make sure that your class structure will work for this game. As you code the project you may need to add some extra data members (instance variables) and functions than those given in the design document. There may be some data and function members you decide not to use.

There is a **project template** on your common drive - Programming\Labs\Project 2**.** It contains the main game loop and loads the font. It also contains some of the header files you will need in the project. You **should** take a copy of it and use it to start your project.

You are being given a **set of images** to use in the game. These image files are to be used to represent the player and enemy sprites etc. in the game. You will find the images on your common drive in the following folder: **Sprites\Project 2**

You are **NOT** allowed to use any other image files.

# It is absolutely **prohibited** to use anyone else’s code in your project. You can ask for help with a particular problem from friends, colleagues, the lecturers in the lab etc but you must code the fix to the problem yourself.

# If you get help from someone or take design/code from the web or elsewhere, you have to ***comment*** in your code to state what help you got and from where.

## AwesomeSquare Gameplay

The player takes the role of an ***AwesomeSquare*** who finds himself (or herself) in a [world](http://en.wikipedia.org/wiki/Maze) made of squares called WorldSquares. If a WorldSquare is kicked, it should move ***one cell*** in the direction that ***AwesomeSquare*** was moving.

The world is inhabited by EvilSquares that are deadly so must be avoided. The EvilSquares move in a random direction. They cannot move into a cell which contains a WorldSquare. They should change direction and try and move into a cell which does not contain a WorldSquare. They could move around in a semi-random way: an EvilSquare may sometimes choose to get away from the AwesomeSquare instead of chasing him (especially on early levels).

The EvilSquares can be **killed** by crushing them with a moving WorldSquare. Only a set number of EvilSquares are on the screen at once e.g. 5 on level 1. A level can be completed by killing all of the EvilSquares. Only **ONE level** is required for basic functionality.

## Basic Functionality:

1. The location of the WorldSquares should be kept in a two dimensional array (12 by 12 array). This array of cells is used to decide where the WorldSquares are drawn on the screen.

Your game should also contain a 1D array of EvilSquares.

1. **Important:** Collision detection between the AwesomeSquare (player) and the WorldSquares and between the EvilSquares and the WorldSquares should be done through using the array of cells. For example the way you will identify that a collision has taken place is by checking if an EvilSquare has moved into the same cell that contains the WorldSquare.
2. Your program should keep track of the player (AwesomeSquare) and its current score. The player should get points for each EvilSquare destroyed. The game should be able to display the current score in the game.
3. When the AwesomeSquare and an EvilSquare meet the AwesomeSquare is killed.
4. The AwesomeSquare movement is through the arrow keys.
5. A WorldSquare is pushed by pressing the space bar.
6. Provide the player with game instructions.
7. Start a new game when the current game is over.

## Summary of some of the Basic Functionality:

**Player AwesomeSquare**

AwesomeSquare moves via the keyboard.

           AwesomeSquare can push/kick blocks.

           AwesomeSquare dies when he meets an EvilSquare.

           AwesomeSquare’s score is recorded.

**WorldSquares**

          The WorldSquare positions are tracked using the 2D array.

           WorldSquares are drawn on the screen.

WorldSquares should move ***one cell*** in the direction the ***AwesomeSquare*** is facing when kicked.

           EvilSquare ‘dies’ if crushed against a WorldSquare.

**EvilSquares**

The EvilSquares should be stored in a 1D array.

            Each should have a movement pattern.

            Should disappear if it crushed with a WorldSquare.

You must use use **Classes** and **Objects** in your Project.

**Demo:** There is a **demo** of the project which you can run to see what a project might look like. See the folder Programming\Labs\Project 2\Project Demo on your common drive. Run the executable file. This demo uses rectangles for the player and evilSquares, you can use images.

## Ideas for Extra Functionality (40%):

* Save an array of items to a text file. For example the array of EvilSquares. You do not need to save all the game objects.
* Read the players’ name from the keyboard and display it on the screen during the game.
* A high score table showing the scores of the games sorted in descending order.
* You could add enemy squares with different behaviours to the game.
* Change the size of the squares and the map.
* Add the ability to rotate squares.
* Add movement by pixels rather than by cells.
* The AwesomeSquare could be able to shoot, or drop SqaureBombs etc.
* You could add a jumping action to AwesomeSquare.
* If a WorldSquare is kicked, it will slide along until it hits another WorldSquare or the edge of the screen. If however, a WorldSquare is already touching something else (in the direction it would slide), it breaks and disappears.
* The EvilSquares can be killed by crushing them with a moving WorldSquare (EvilSquares do not stop the WorldSquare so more than one EvilSquare can be killed with one kick).
* Different levels in the game. Each level could have a different maze and level of difficulty.
* The game continues with different coloured EvilSquares in a new maze. The number of EvilSquares on screen, the number of 'reserve' EvilSquares , the speed of the EvilSquares and the intelligence of the EvilSquares all increase on each successive level, making it much more difficult to successfully line up the EvilSquares.

## Code

Your project must be **commented appropriately**: project comment, function comments and comments should be used to explain pieces of code though out the functions where necessary.

Your project should have good structure and layout. There should be appropriate use of classes and objects, appropriate use of arrays & functions, parameter passing, indentation, choice of

## Core and Games Mark

40% of the games mark is for extra functionality.

The **core mark does not require** any extra functionality to get 100%.

## IMPORTANT:

You're required to demonstrate the project in class and to submit it to the M drive. **If you do not demonstrate the project in class you will NOT receive any marks for the project. Late entries will not be considered.**

There will be a **written exam based on the basic functionality** of the project, it will NOT include any extra functionality. You need to pass this exam in order to be allocated your demo project mark. If there is a considerable difference between your written exam mark and your demo core mark, your demo mark may be reduced downwards. You must sit the written exam in order for your project demo mark to be allocated.

**This project (design & code) is worth 20% of your final mark in Programming.**

## Submission Dates

**You will be asked to demo parts of your project working before the final demo. This is the minimum you should have done on these dates.**

**Monday 12th of March demo** Draw the maze. The player should move around within the boundary of the maze using the 4 keys. The player’s image should change as he changes direction.

**Monday 19th of March demo** The array of evilSquares should be moving around within the boundary of the maze. Collision detection between the player and evilSquare objects.

**Monday 09th of April demo** The player can kick a worldSq one cell and crush an evilSquare & scoring.

Final Project code due: **Wednesday the 11thof April (put on your M drive).**

**Final demos are taking place Monday 16th of April in the lab.**

## Saving Your Project

**Important: Your program should not contain any compile time errors when you submit it to the M drive.**

Within the Programming folder on your M: exam directory, create a new folder called Project2StudentName. Copy the project to this folder. N.B Put your name on the folder i.e. Project2MarkRyan.

**Also sign the Plagiarism declaration** (It is in the project folder of your common drive) and copy it to your project folder.

**Marking Scheme**

|  |  |  |
| --- | --- | --- |
|  | **Games** | **Core** |
| **Correctness** | 40% | 80% |
| **Code Quality** | 20% | 20% |
| **Extra Functionality** | 40% |  |
| **Total** | **100%** | **100%** |