World Defence:

Technical Manual

Version 1.0

18/04/2015

Created By: Siobhan Wallace

Table of Contents

[Brief 3](#_Toc418173289)

[System Requirements 3](#_Toc418173290)

[Hardware Requirements 3](#_Toc418173291)

[Software Requirements 3](#_Toc418173292)

[Functionality 4](#_Toc418173293)

[Implementation Comments 4](#_Toc418173294)

[Additional Functionality 5](#_Toc418173295)

[Removed functionality 5](#_Toc418173296)

[Naming Conventions 5](#_Toc418173297)

[Development Environment 5](#_Toc418173298)

[Design 6](#_Toc418173299)

[Classes 6](#_Toc418173300)

[Updated Class Diagram 19](#_Toc418173301)

[External Libraries 19](#_Toc418173302)

[High-Score Database 20](#_Toc418173303)

[ERD 20](#_Toc418173304)

[Database Design 20](#_Toc418173305)

[Score 20](#_Toc418173306)

[Player 20](#_Toc418173307)

[SQL Commands 21](#_Toc418173308)

[Testing 21](#_Toc418173309)

[Testing Methodology 21](#_Toc418173310)

[Test Plan 21](#_Toc418173311)

[Testing Cases 22](#_Toc418173312)

[Test Player Feedback 25](#_Toc418173313)

[Appendix 25](#_Toc418173314)

[Data Dictionary 25](#_Toc418173315)

[Full Code 33](#_Toc418173316)

## Brief

As part of my HND Software Development: Year Two Graded Unit, I propose to create a 2D arcade game. This game will be based off the 1978 classic ‘Space Invaders’, and in which the user may defeat waves of enemies by controlling a turret.

This game will be written in the C# programming language, using MS Visual Studio as the development environment, with the addition of Microsoft’s XNA framework.

As the proposed game will be of the Fixed Shooter genre, it will be following these basic rules;

1. The player must defeat waves of mobs in order to advance to the next level
2. The player is defeated if they either fail to stop the mobs or are killed.

## System Requirements

The game UI will be designed around keyboard input, and all graphical elements should be reasonably comfortable to use.

The game has a simple design and as such should run on any modern pc. A keyboard and mouse is required in order to fully interact with the game.

### Hardware Requirements

|  |  |  |
| --- | --- | --- |
|  | Minimum | Recommended |
| **Operating System:** | Windows XP with SP2 | Windows 7 or 8 |
| **Processor:** | 1GHz |  |
| **Graphics Card:** | Support for Shader Model 1.1 | Card with 512mb or better |
| **Memory:** | 512MB | 1GB or better |
| **Sound Card:** | N/A | Standard card or better |
| **Input:** | Keyboard | Keyboard / Xbox Controller |

Software Requirements

As World Defence uses Microsoft’s XNA framework, it requires either XNA or its redistributable to be installed on the client machine. The installer for World Defence automatically installs this for the user.  
In order for the high score database to function, either Microsoft Access or Microsoft Access Runtime must be installed.

Microsoft Access Runtime can be found here: <http://www.microsoft.com/en-gb/download/details.aspx?id=39358>

## Functionality

|  |  |  |
| --- | --- | --- |
| Functional Requirements | | |
| Number | **Description** | **Implemented** |
| 1 | The Player will be shown game menu on application start | **Yes** |
| 2 | The Player may exit the application at any time | **Yes** |
| 3 | The Player may quit the game at any time | **Yes** |
| 4 | The Player may pause the game at any time | **Yes** |
| 5 | The Player may exit to the main menu at any time | **Yes** |
| 6 | When the Player chooses to exit, a confirmation window will appear | **Yes** |
| 7 | The Player will “die” if hit 3 times during the game |  |
| 8 | The Player must defeat 10 ‘waves’ of enemies |  |
| 9 | The Boss will spawn after the 10th ‘wave’ of enemies |  |
| 10 | The Player will be able to toggle game sound on/off | **Yes** |
| 11 | Music will be playing when Game is running | **Yes** |
| 12 | Visual and Auditory effects will play when the Player is hit. | **Partially** |
| 13 | The Player will be prompted to enter their name on Game Over |  |

|  |  |  |
| --- | --- | --- |
| Non-Functional Requirements | | |
| Number | **Description** | **Implemented** |
| 14 | The Game will run on the Windows platform, with support primarily being aimed at Windows 7 and 8. | **Yes** |
| 15 | Game background will be black. | **No** |
| 16 | The High Score will only display top 10 entries |  |
| 17 | The Player will be awarded points for each normal enemy eliminated | **Yes** |
| 18 | The Player will be awarded 100 points for eliminating the Boss |  |
| 19 | Each enemy will come in one of the following colours (red, purple, yellow, orange, green) | **Partially** |

### Implementation Comments

|  |  |
| --- | --- |
| Number | Reason |
| 12 | Sound will play when player is defeated, and lives are tracked on screen |
| 15 | A minimalistic background was implemented instead of a plain black background, as it was decided that it makes the game look better. |
| 19 | Enemies come in 5 different types, and each type is a different colour, which aids the player in identification. It was decided that additional colour differences would only confuse the player |

### Additional Functionality

* Added ability for player to change their texture.

### Removed functionality

* Removed VS Mode due to time constraints and complexity.
* Removed player shields & replaced with lives tracker due to time constraints
* Removed bonus enemy that would periodically appear due to unnecessary complexity it added with the update code for enemies

## Naming Conventions

In accordance with the naming conventions set out by “Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition”, Pascal Case has been used for all classes and method names, whilst Camel Case was used for all variable names.

The naming conventions followed can be found here: <https://msdn.microsoft.com/en-us/library/ms229045.aspx>

## Development Environment

The native environment used to develop World Defence was Visual Studio Professional 2013, as Siobhan is fairly familiar with this environment

## Design

### Classes

|  |  |  |
| --- | --- | --- |
| Class Name | Player | |
| Use | Holds and initialises all variables for the player | |
| Initial Design | “Holds and initialises all variables for the player” | |
| Changes from Initial Design | Consolidated Get/SetPlayerCoords into GetPCoords(), changed playerCoords to Vector2  Removed playerColour, replaced with playerSprite  Separated movePlayer() into MovePlayerLeft() & MovePlayerRight()  Added methods for boundary collision  Added constructor | |
| Variables | **Name** | **Use** |
| playerCoords | Holds the current position of the player |
| movement | Holds the current movement of the player |
| screenBoundary | Holds boundary of the screen to detect collision |
| isAlive | Stores if player is alive; to determine game over |
| playerShield | Holds the players current shield level, between 3 and 0 |
| playerScore | Holds the players current score |
| playerSprite | Texture for the player |
| playerSpeed | Holds the player speed |
| Methods | **Name** | **Use** |
| GetPCoords() | Gets the players current position on screen |
| AddScore() | Sets the players current score |
| GetBoundary() | Gets the players boundary to detect collision |
| Player() | Constructor for the player; sets initial values |
| MovePlayerLeft() | Handles moving the player left |
| MovePlayerRight() | Handles moving the player right |
| ScrBoundaryCheck() | Checks if the player has left the screen boundaries |
| Draw() | Draws the player on screen |

|  |  |  |
| --- | --- | --- |
| Class Name | Lasers | |
| Use | Holds and initialises all variables for the lasers | |
| Initial Design | “Holds and initialises all variables for the lasers” | |
| Changes from Initial Design | Removed isFriendly, not required  Consolidated (Move)LaserCoordinates into UpdatePosition(), changed from point to Vector2  Added boundary detection  Added speed variable  Added update logic  Added active detection  Changed laserColour to changing the texture colour, rather than redrawing the entire laser | |
| Variables | **Name** | **Use** |
| boundary | Holds the boundary |
| isActive | Holds the current state of the laser |
| laserPos | Holds the current position of the laser |
| laserSpeed | Holds the speed of the laser |
| playerLaserTex | Holds the texture for the player laser |
| movement | Holds the lasers current movement axis |
| screenBoundary | Holds the screen bounds |
| Methods | **Name** | **Use** |
| Draw() | Draws the lasers |
| Fire() | Handles the firing of the laser |
| HasExited() | Detects if the laser has left the top of the screen |
| IfIsActive() | Returns the current state of the laser |
| Lasers() | Constructor for the lasers |
| SetIsActive() | Sets the laser to be active or not |
| SetLaserColour() | Sets the laser colour |
| Update() | Handles and executes all update logic |
| UpdatePosition() | Updates the laser’s current position |

|  |  |  |
| --- | --- | --- |
| Class Name | Enemies | |
| Use | Holds the variables required & used by the enemies in-game | |
| Initial Design | “Holds the variables required & used by the enemies in-game” | |
| Changes from Initial Design | Changed enemyCoordinates to enemyPos, changed from Point to Vector2.  Removed size & hp, not needed, replaced with scoreValue & isAlive  Removed enemyColour, not needed, replaced with varying enemySprites  Relocated moveEnemies() to game modes  Added collision detection | |
| Variables | **Name** | **Use** |
| enemyPos | Holds the enemies current position |
| enemySprite | Holds the enemy sprite |
| isAlive | Holds the current state of the enemy |
| scoreValue | Holds the score value of the enemy |
| Methods | **Name** | **Use** |
| CollisionCheck() | Checks if the enemies have collided with the player laser |
| Draw() | Draws the enemies |
| Enemies() | Constructor for the enemies |
| GetIsAlive() | Gets if the enemies are currently alive |

|  |  |  |
| --- | --- | --- |
| Class Name | PlayerIndexEventArgs\* | |
| Use | Handles player index events, allows game to handle more than one player input. Inherits from EventArgs() | |
| Initial Design | Was not in the original design | |
| Changes from Initial Design | N/A | |
| Variables | **Name** | **Use** |
| playerIndex | Holds the index of each player |
| Methods | **Name** | **Use** |
| PlayerIndexEventArgs() | Gets the index of the player who triggered the event |

|  |  |  |
| --- | --- | --- |
| Class Name | ArcadeGameplay | |
| Use | Holds all the variables and logic required in order to run the arcade game mode | |
| Initial Design | Was not in the original design | |
| Changes from Initial Design | N/A | |
| Variables | **Name** | **Use** |
| bgTex | Holds the game background texture |
| content | Loads a new content manager for the menu content |
| enemies | Loads a new enemies for handling enemy attributes |
| enemy1Sprite | Holds the enemy texture, variation 1 |
| enemy2Sprite | Holds the enemy texture, variation 2 |
| enemy3Sprite | Holds the enemy texture, variation 3 |
| enemy4Sprite | Holds the enemy texture, variation 4 |
| enemyHeight | Holds the height of the enemies |
| enemyLasers | Creates a new lasers to handle enemy lasers |
| enemyLaserSound | Holds the sound used for the enemies firing their laser |
| enemyLaserTex | Holds the texture for the enemies laser |
| enemyWidth | Holds the width of the enemy |
| fireDelay | Holds how long the player must wait before being able to fire the laser again |
| gameFont | Holds the sprite font used by game |
| gameMusic | Holds the music used in game |
| gameMusicInstance | Holds the instance of the game music |
| playerLaserTex | Holds the texture of the player laser |
| pauseAlpha | Holds the alpha value for pausing the screen (used for fading to black) |
| player | Loads a new player for handling the player attributes |
| playerLasers | Creates a new lasers to handle the player lasers |
| playerlaserSound | Holds the sound used for the player firing their laser |
| prevFiretime | Holds the last time the player fired the laser |
| scoreValue | Holds the score value of the enemies |
| screenBoundary | Hold the current screen bounds |
| Methods | **Name** | **Use** |
| AddEnemy() | Adds the enemies via an array |
| AddLaser() | Adds the player lasers |
| ArcadeGamePlay() | Constructor |
| Draw() | Draws the in-game components |
| HandleInput() | Allows the game to handle player input |
| Initialize() | Initialises all variables |
| LoadContent() | Loads the content for this screen |
| PlaySound() | Plays the game music |
| UnloadContent() | Unloads the content for this screen |
| Update() | Handles and executes all update logic |
| UpdateCollision() | Updates the collision for the enemy and player |
| UpdateEnemies() | Updates the enemy attributes |

|  |  |  |
| --- | --- | --- |
| Class Name | EndlessGameplay | |
| Use | Holds all the variables and logic required in order to run the endless game mode | |
| Initial Design | Was not in the original design | |
| Changes from Initial Design | N/A | |
| Variables | **Name** | **Use** |
| bgTex | Holds the game background texture |
| content | Loads a new content manager for the menu content |
| enemies | Loads a new enemies for handling enemy attributes |
| enemy1Sprite | Holds the enemy texture, variation 1 |
| enemy2Sprite | Holds the enemy texture, variation 2 |
| enemy3Sprite | Holds the enemy texture, variation 3 |
| enemy4Sprite | Holds the enemy texture, variation 4 |
| enemyHeight | Holds the enemies height |
| enemyLasers | Creates a new lasers to handle enemy lasers |
| enemyLaserSound | Holds the sound used for the enemies firing their lasers |
| enemyLaserTex | Holds the texture used for the enemies laser |
| enemyWidth | Holds the width of the enemies |
| fireDelay | Holds how long the player must wait before being able to fire the laser again |
| gameFont | Holds the sprite font used by the game |
| gameMusic | Holds the game music |
| gameMusicInstance | Holds the game music instance |
| playerLaserTex | Holds the texture used for the player firing their laser |
| pauseAlpha | Holds the alpha value for pausing the screen (for fading to black) |
| player | Loads a new player for handling player attributes |
| playerLasers | Creates a new lasers to handle the player lasers |
| playerlaserSound | Holds the sound for the player firing their laser |
| prevFiretime | Holds the last time the player fired the laser |
| scoreValue | Holds the score value of the enemies |
| screenBoundary | Holds the current screen bounds |
| Methods | **Name** | **Use** |
| AddEnemy() | Adds the enemies via array |
| AddLaser | Adds the player lasers |
| EndlessGamePlay() | Constructor for the game mode |
| Draw() | Draws the in-game components |
| HandleInput() | Allows the game to handle player input |
| Initialize() | Initialises all variables |
| LoadContent() | Loads the content for this screen |
| PlaySound() | Plays the game music |
| UnloadContent() | Unloads the content for this screen |
| Update() | Handles and executes all update logic |
| UpdateCollision() | Updates the collision between player and enemies |
| UpdateEnemies() | Dynamically updates the enemy attributes |

|  |  |  |
| --- | --- | --- |
| Class Name | CoopGameplay | |
| Use | Holds all the variables and logic required in order to run the co-operative game mode | |
| Initial Design | Was not in the original design | |
| Changes from Initial Design | N/A | |
| Variables | **Name** | **Use** |
| bgTex | Holds the background texture of the game |
| bossCurrentHP | Holds the boss’s current hp |
| bossLaserTex | Holds the texture of the boss’ laser |
| bossPos | Holds the boss current position |
| bossSprite | Holds the texture of the boss |
| content | Loads a new content manager for the menu content |
| enemies | Creates a new instance of the enemies() class |
| enemy1Sprite | Holds the enemy texture, variation 1 |
| enemy2Sprite | Holds the enemy texture, variation 2 |
| enemy3Sprite | Holds the enemy texture, variation 3 |
| enemy4Sprite | Holds the enemy texture, variation 4 |
| enemyHeight | Holds the enemies height |
| enemyLasers | Creates a new lasers to handle the enemy lasers |
| enemyLaserSound | Holds the sound for the enemies firing their laser |
| enemyLaserTex | Holds the texture of the enemy laser |
| enemyWidth | Holds the width of the enemies |
| explosions | Holds the sound for player death |
| fireDelay | Holds how long the player must wait before being able to fire the laser again |
| gameFont | Holds the font used by the game |
| gameMusic | Holds the game music |
| gameMusicInstance | Holds the game music, with added functionality (playing, paused etc.) |
| playerLaserTex | Holds the texture for the player laser |
| pauseAlpha | Holds the alpha value for pausing the screen (used for fading to black) |
| player | Loads a new player for handling player attributes |
| player2 | Loads a new player for handling player 2 attributes |
| player2Pos | Holds the current position of player2 |
| player2Sprite | Holds the texture of player 2 |
| playerLasers | Creates a new lasers for handling player lasers |
| playerLaserSound | Holds the sound of the player firing their laser |
| prevFireTime | Holds the last time the player fired the laser |
| scoreValue | Holds the score value of the enemies |
| screenBoundary | Holds the screen bounds |
| Methods | **Name** | **Use** |
| AddEnemy() | Adds the enemies via array |
| AddLaser() | Adds the lasers |
| Draw | Draws the in game components |
| CoopGameplay() | Constructor for this game mode |
| HandleInput() | Allows the game to handle player input |
| Initialize() | Initialises all variables |
| LoadContent() | loads the content for this screen |
| PlaySound() | Plays the game music |
| UnloadContent() | Unloads the content for this screen |
| Update() | Handles & executes all update logic |
| UpdateBoss() | Dynamically updates the boss |
| UpdateEnemies() | Dynamically updates the enemies |
| UpdateCollision() | Update the collision between the player and enemies |

|  |  |  |
| --- | --- | --- |
| Class Name | GameScreen\* | |
| Use | Holds all the generic attributes for loading the game screens and acts as the parent class for each screen | |
| Initial Design | Was not in the original design | |
| Changes from Initial Design | N/A | |
| Variables | **Name** | **Use** |
| conPlayer | Holds the index of the player who triggered the events |
| isExiting |  |
| isPopup |  |
| otherScreenHasFocus |  |
| screenManager |  |
| screenState |  |
| transOffTime |  |
| transOnTime |  |
| transPos |  |
| Methods | **Name** | **Use** |
| ConPlayer() | Gets the index of the player who triggered events |
| IsActive() |  |
| IsExiting() |  |
| IsPopup() |  |
| ScreenManager() |  |
| ScreenState() |  |
| TransAlpha() |  |
| TransOffTime() |  |
| TransOnTime() |  |
| TransPos() |  |

|  |  |  |
| --- | --- | --- |
| Class Name | InputState\* | |
| Use |  | |
| Initial Design | Was not in the original design | |
| Changes from Initial Design | N/A | |
| Variables | **Name** | **Use** |
| currentGamePadStates |  |
| currentKeyboardStates |  |
| gamePadWasConnected |  |
| lastGamePadStates |  |
| lastKeyboardStates |  |
| maxInputs |  |
| Methods | **Name** | **Use** |
| InputState() |  |
| IsFired() |  |
| IsMenuCancel() |  |
| IsMenuDown() |  |
| IsMenuUp() |  |
| IsMenuSelect() |  |
| IsNewButtonPress() |  |
| IsNewKeyPress() |  |
| IsPauseGame() |  |
| Update() |  |

|  |  |  |
| --- | --- | --- |
| Class Name | ScreenManager\* | |
| Use |  | |
| Initial Design | Was not in the original design | |
| Changes from Initial Design | N/A | |
| Variables | **Name** | **Use** |
| blankTex | Holds the blank texture used in the fadetoblackbuffer |
| gameFont | Holds the game font used in game |
| input | Creates a new InputState to handle this screen |
| isInitialized | Holds if the current screen is initialised or not |
| sBatch | Holds the spritebatch used for drawing in game |
| screens | Creates a new list using the gamescreen class |
| screensToUpdate | Creates a new list using the gamescreen class |
| Methods | **Name** | **Use** |
| AddScreen() |  |
| Draw() |  |
| FadeBackBufferToBlack() |  |
| GetScreens(Initialize()) |  |
| LoadContent() | Loads content for this screen |
| RemoveScreen() |  |
| ScreenManager() | Constructor for the screen manager |
| UnloadContent() | Unloads content for this screen |
| Update() |  |

|  |  |  |
| --- | --- | --- |
| Class Name | HighscoreScreen | |
| Use | The screen shown to the player when they view high scores, it acts as the interface for the player that wishes to view scores | |
| Initial Design | Was originally View\_High\_Score() | |
| Changes from Initial Design | Added variables, relocated DisplayHighscores/UpdateHighscores() to DatabaseInt  Added option to choose which game mode to display for | |
| Variables | **Name** | **Use** |
| currentMode | Holds which mode is currently selected |
| dataInt | Creates a new DataInt to handle interfacing with the database |
| gameMode | Holds the currently selected game mode for queries |
| maxRows | Holds the maximum number of rows that will be drawn |
| modeEntry | Creates a new menu entry to hold the mode option |
| rowDraw | Holds the number of rows in the current query |
| Methods | **Name** | **Use** |
| HighscoreScreen() | Constructor for this class |
| LoadContent() | Loads content for this screen |
| ModeEntrySel() | Event handler for when user selects an option |
| SetMenuText() | Sets the text of the menu |

|  |  |  |
| --- | --- | --- |
| Class Name | Loading\* | |
| Use | Holds all the variables required for the loading screen and acts as the loading screen | |
| Initial Design | Was not in the original design | |
| Changes from Initial Design | N/A | |
| Variables | **Name** | **Use** |
| loadingIsSlow | Holds if loading is taking a while, which determines if the loading message should be shown |
| otherScreensGone | Holds if the other screens have transitioned off |
| screensToLoad | Holds which screens are to be loaded |
| Methods | **Name** | **Use** |
| Draw() | Draws the loading screen |
| Load() | Activates the loading screen |
| Loading() | Constructor for the loading screen |
| Update() | Updates the loading screen |

|  |  |  |
| --- | --- | --- |
| Class Name | MainMenu | |
| Use | Holds all the variables required and acts as the main menu screen | |
| Initial Design | Was not in the original design | |
| Changes from Initial Design | N/A | |
| Variables | **Name** | **Use** |
| None | N/A |
| Methods | **Name** | **Use** |
| MainMenu() | Constructor for the main menu |
| EndlessModeMenuEntrySel() | Event handler for when endless Mode entry is selected |
| ArcadeModeMenuEntrySel() | Event handler for when arcade Mode entry is selected |
| CoopModeMenuEntrySel() | Event handler for when co-op Mode entry is selected |
| OptionsMenuEntrySel() | Event handler for when options entry is selected |
| HighscoreMenuEntrySel() | Event handler for when high score entry is selected |
| OnCancel() | When user wants to exit, asks for confirmation |
| ConfirmExitMessageBoxAccepted() | Event handler for when user selects ok on message box |

|  |  |  |
| --- | --- | --- |
| Class Name | MenuEntry\* | |
| Use | Holds the attributes needed for the menu entries and acts as the parent class to create new menu entries | |
| Initial Design | Was not in the original design | |
| Changes from Initial Design | N/A | |
| Variables | **Name** | **Use** |
| pos | Holds the position of the menu entries |
| selFade | Holds the value for selection fading |
| text | Holds the text in the enu entries |
| Methods | **Name** | **Use** |
| Draw() | Draws the menu entries |
| GetHeight() | Gets the height of the menu entries |
| GetWidth() | Gets the width of the menu entries |
| MenuEntry() | Constructor for this class |
| OnSelectEntry() |  |
| Update() | Handles and executes all update logic |

|  |  |  |
| --- | --- | --- |
| Class Name | MenuScreen\* | |
| Use | Holds all the generic variables required to load each menu screen and acts as the parent class for each menu | |
| Initial Design | Was not in the original design | |
| Changes from Initial Design | N/A | |
| Variables | **Name** | **Use** |
| content | Loads a new content manager for the menu content |
| menuEntries | Creates menu entries as a new menu entry list |
| menuMusic | Holds the menu music |
| menuMusicInstance | Holds the menu music |
| menuTitle | Holds the menu title |
| selEntry | Holds the integer for entry selection |
| Methods | **Name** | **Use** |
| Draw() | Draws the menu screens |
| HandleInput() | Handles player input for the menu screens |
| LoadContent() | Loads content for the menu screens |
| MenuScreen() | Constructor for the menu screen |
| Update() | Updates the menu screens |
| UpdateMenuEntries() | Updates the menu entries |
| OnSelectEntry() | Event handler for when the player selects a menu entry |
| OnCancel() | Event handler for when the player selects cancel |

|  |  |  |
| --- | --- | --- |
| Class Name | MsgboxScreen\* | |
| Use | Holds all the variables associated with the message box and acts as the message box screen | |
| Initial Design | Was not in the original design | |
| Changes from Initial Design | N/A | |
| Variables | **Name** | **Use** |
| message | Holds the message to be displayed |
| gradientTexture | Holds the texture for the background of the message box |
| Methods | **Name** | **Use** |
| Draw() | Draws the message box on screen |
| HandleInput() | Accepts user input for the message box |
| LoadContent() | Loads the screen content |
| MsgboxScreen() | The constructor for the message box |

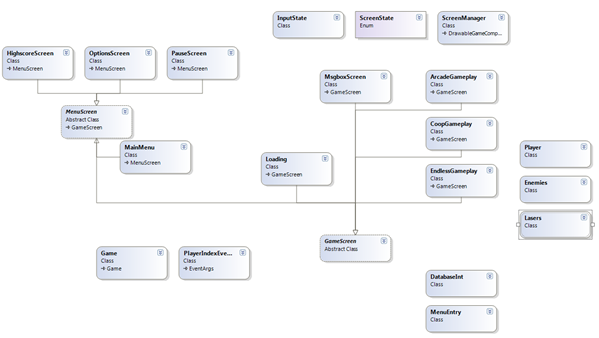
|  |  |  |
| --- | --- | --- |
| Class Name | OptionsScreen | |
| Use | Holds all the variables associated with the options menu | |
| Initial Design | “Holds all the variables associated with the options menu” | |
| Changes from Initial Design | Removed hex colour due to difficulty of coding, added more variables, renamed original variables to better fit. | |
| Variables | **Name** | **Use** |
| currentColour | Holds the currently shown colour option |
| currentMusic | Holds the currently shown music option |
| currentShip | Holds the currently shown ship option |
| laserColourEntry | Initialises laserColour as a MenuEntry class |
| musicEntry | Initialises music as a MenuEntry class |
| playerShipEntry | Initialises playership as a MenuEntry class |
| Methods | **Name** | **Use** |
| LaserColourEntrySel() | Toggles through the laser colour options when selected |
| MusicEntrySel() | toggles the music on or off when selecte |
| PlayerShipEntrySel() | Toggles through the player ship options when selected |
| LoadContent() | Loads the content for this menu |
| OptionsScreen() | Constructor for the options screen |
| SetMenuText() | Sets the menu texts |

|  |  |  |
| --- | --- | --- |
| Class Name | PauseScreen\* | |
| Use | Holds all the variables associated with the pause screen and acts as the games pause screen | |
| Initial Design | Was not in the original design | |
| Changes from Initial Design | N/A | |
| Variables | **Name** | **Use** |
| None | N/A |
| Methods | **Name** | **Use** |
| PauseScreen() | Constructor for the pause screen |
| ConfirmMsgBoxAccepted() | Handles the user selecting quit game event |
| QuitGameMenuEntrySel() | Handles user confirming quit event |

|  |  |  |
| --- | --- | --- |
| Class Name | DatabaseInt | |
| Use | Holds all the variables required for interacting with the database and acts as the games link to the database | |
| Initial Design | “acts as the games link to the database”  Was originally Database\_Int() | |
| Changes from Initial Design | Added database connectivity  Separated Write\_Database() into WriteDatabase() and AddNameDB()  Added Draw()  Added database checks | |
| Variables | **Name** | **Use** |
| connection | Initialises a new connection |
| conString | Tells the game where to look for the database |
| dataRowC | Initialises the datarow collection |
| dataSet | Initialises the dataset |
| playerID | Holds the playerID |
| rowDraw | Tells the game how many rows to display |
| transaction | Holds the initial transaction value |
| Methods | **Name** | **Use** |
| AddNameDB() | Adds player name to database |
| CheckRows() | Checks how many rows are currently in the database |
| DataBaseInt() |  |
| Draw() | Draws the database rows |
| ReadDatabase() | Reads the content from the database |
| WriteDatabase() | Writes the high score to the database |

|  |  |  |
| --- | --- | --- |
| Class Name | Game | |
| Use | The main class of the game | |
| Initial Design | “the entry point of the game” was originally Game\_Screen() | |
| Changes from Initial Design | Reworked class, most original components now part of game mode screens etc.  Removed sound variable, relocated to menu system/game modes | |
| Variables | **Name** | **Use** |
| graphics | Creates a new instance of GraphicsDeviceManager to handle graphics devices |
| preloadAssets | Preloads assets to reduce strain |
| sBatch | Holds the spritebatch used for drawing in game |
| screenManager | Creates a new screenManger for handling this screen |
| Methods | **Name** | **Use** |
| Draw() | Draws the assets for this class |
| Game() | Constructor for the class |
| Initialise() | Initialises variables used by this class |
| LoadContent() | Loads the content for this class |
| UnloadContent() | Unloads the content for this class |
| Update() | Handles and executes all update logic |

### Updated Class Diagram



### External Libraries

Microsoft’s s XNA was the main library used in the creation of this game, due to the many features of XNA that makes game creation an easier task.

Additionally, whilst researching the possible methods for screen management, a code sample/tutorial designed by Microsoft, detailing a possible method of game state management, was found at the following location:

<http://xbox.create.msdn.com/en-US/education/catalog/sample/game_state_management>

Any classes that pertain to, partially or wholly, the above code sample have been **marked with an asterisk \*,** although the original samples used may have been edited.

All classes marked with \* have the following copyright:

// Microsoft XNA Community Game Platform

// Copyright (C) Microsoft Corporation. All rights reserved.

**The Microsoft Permissive License (Ms-PL) allows the editing and redistribution of any contributions made to the sample, as long as the copyright is credited.**

## High-Score Database

### ERD

|  |  |
| --- | --- |
|  | # - indicates primary ley  \* - indicates a not null value    Indicates a mandatory one to many (1:M) relationship |
| Player MUST have one or more than one Score  Score MUST have one and only one Player | |

### Database Design

### Score

Primary Key: Rank

Foreign Key: player\_ID

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Attribute Name | Database name | Optionality | Data type | Description |
|  | Rank | Not Null | Int(2) | Stores the rank achieved by the player  (between 1-10) |
|  | Player\_ID | Not Null | Int(4) | Stores the unique number assigned to each player |
|  | Score | Not Null | Int(16) | Stores the score achieved by the player |
|  | Mode | Not Null | String | Stores which game mode was played |

### Player

Primary Key: player\_ID

Foreign Key: N/A

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Attribute Name | Database name | Optionality | Data type | Description |
|  | Player\_ID | Not Null | Int(4) | Stores the unique number assigned to each player |
|  | Name | Not Null | String | Stores the player name |

### SQL Commands

|  |  |
| --- | --- |
|  |  |
| SQL Command | **Use** |
|  |  |
|  |  |
|  |  |
|  |  |

## Testing

### Testing Methodology

Testing for World Defence was split between white-box and black-box testing. Due to constraints on time and limited resources available, the black-box testing was done on very few machines. The machine it was coded on, a desktop which runs Windows 7 Ultimate, 64bit, was used for this purpose.

In an ideal situation, testing would be conducted more thoroughly, using many other machines to fully test compatibility.

Test cases have been designed to test functionality, with the expected and actual results being documented. Any bugs found will be dealt with before another test session takes place to note if the problems have been resolved.

### Test Plan

Testing will be performed on the major areas of the game: The Menu System, the High score System and the three Game Modes. Due to time constraints, Integration testing will be done for the UI, taking in the menu system as a whole, rather than individual unit tests per screen. However, Unit testing will be conducted for each Game Mode, and the Database Integration, with each Unit being taken as the class, due to the game being object oriented.

An example testing case form is shown below;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| System Testing | | | | |
| Test | **Expected Result** | **Actual Result** | **Pass/Fail** | **Comments** |
|  |  |  |  |  |
|  |  |  |  |  |

### Testing Cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Menu System Testing | | | | |
| Test – main screen | **Expected Result** | **Actual Result** | **Pass/Fail** | **Comments** |
| Can the user access the high score menu? | Player can access high score menu |  |  |  |
| Can the user access the options menu? | Player can access options menu |  |  |  |
| Can the user access the endless game mode? | Player can access endless mode |  |  |  |
| Can the user access the co-op game mode? | Player can access co-op mode |  |  |  |
| Can the user access the arcade game mode? | Player can access arcade mode |  |  |  |
| Test – Pause Menu | | | | |
| Can the user pause the game at any time? | Player can pause at any time |  |  |  |
| Can the user quit the game from the pause menu? | Player can quit from pause screen |  |  |  |
| Can the user return to the main menu? | Player can return to main menu |  |  |  |
| Test – Options Menu | | | | |
| Can the user change their laser colour? | Player can change laser colour |  |  |  |
| Can the user change their model? | Player can change model |  |  |  |
| Can the user toggle sound on/off? | Player can toggle sound |  |  |  |
| Test – All Menu’s | | | | |
| Can the user return to the main menu from any of the above screens? | Player can return to the main menu |  |  |  |
| Does the menu music correctly play in each of the above menus? | Menu music plays correctly |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| High score System Testing | | | | |
| Test | **Expected Result** | **Actual Result** | **Pass/Fail** | **Comments** |
| Can player enter their name? | Player can enter their name |  |  |  |
| Can the player choose which game scores to display? | Player can choose which game mode they would like to have scores displayed for |  |  |  |
| Does the game correctly retrieve the scores? | Game correctly shows score table |  |  |  |
| Does the database store more than 10 entries? | Database does not store more than 10 entries |  |  |  |
| Does the game correctly show which rank was achieved? | Game correctly shows achieved rank |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Endless Mode Testing | | | | |
| Test | **Expected Result** | **Actual Result** | **Pass/Fail** | **Comments** |
| Can the player exit the game-screen boundary? | Player cannot exit the game-screen boundary. |  |  |  |
| Does the game end if the player’s health drops to 0? | Game ends if player health drops to 0 |  |  |  |
| Does the game end if the player fails to stop the enemies? | Game ends if player fails to stop enemies |  |  |  |
| Does the boss spawn after every 10 waves? | Boss spawns after every 10 waves |  |  |  |
| Does the boss grant the correct number of points when killed? | Boss grants correct amount of points (100) |  |  |  |
| Do the enemies award the correct number of points when killed? | Enemies award correct number of points when killed |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Co-Op Mode Testing | | | | |
| Test | **Expected Result** | **Actual Result** | **Pass/Fail** | **Comments** |
| Can the players exit the game-screen boundary? | Players cannot exit the game-screen boundary. |  |  |  |
| Can each player move independently of the other? | Players can move independently of each other |  |  |  |
| Does the game end if any player dies? | Game ends if any player dies |  |  |  |
| Does the game end if the players fail to stop the enemies? | Game ends if players fail to stop enemies |  |  |  |
| Does the game end if the players defeat the boss? | Game ends if players defeat boss |  |  |  |
| Do the enemies award the correct number of points when killed? | Enemies award correct number of points when killed |  |  |  |
| Does the boss grant the correct number of points when killed? | Boss award correct number of points when killed (100) |  |  |  |
| Does the boss spawn after the 10th wave of enemies? | Boss spawns after the 10th wave of enemies |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Arcade Mode Testing | | | | |
| Test | **Expected Result** | **Actual Result** | **Pass/Fail** | **Comments** |
| Can the player exit the game-screen boundary? | Player cannot exit the game-screen boundary. |  |  |  |
| Does the game end if the player’s health drops to 0? | Game ends if player health drops to 0 |  |  |  |
| Does the game end if the player fails to stop the enemies? | Game ends if player fails to stop enemies |  |  |  |
| Do the enemies award the correct number of points when killed? | Enemies award correct number of points when killed |  |  |  |
|  |  |  |  |  |

## Test Player Feedback

## Appendix

### Data Dictionary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Attribute Name | Data Type | Accessibility | Location | Description |
| playerCoords | Vector2 | Public | Player() | Holds the current position of the player |
| movement | Vector2 | Private | Player() | Holds the current movement of the player |
| screenBoundary | Rectangle | Private | Player() | Holds boundary of the screen to detect collision |
| isAlive | Boolean | Private | Player() | Stores if player is alive; to determine game over |
| playerShield | Int | Private | Player() | Holds the players current shield level, between 3 and 0 |
| playerScore | Int | Private | Player() | Holds the players current score |
| playerSprite | Texture2D | Private | Player() | Texture for the player |
| playerSpeed | Float | Private | Player() | Holds the player speed |
|  | | | | |
| boundary | Rectangle | Private | Lasers() | Holds the boundary |
| isActive | Boolean | Public | Lasers() | Holds the current state of the laser |
| laserPos | Vector2 | Public | Lasers() | Holds the current position of the laser |
| laserSpeed | Float | Private | Lasers() | Holds the speed of the laser |
| playerLaserTex | Texture2D | Private | Lasers() | Holds the texture for the player laser |
| movement | Vector2 | Private | Lasers() | Holds the lasers current movement axis |
| screenBoundary | Rectangle | Private | Lasers() | Holds the screen bounds |
|  | | | | |
| enemyPos | Vector2 | Public | Enemies() | Holds the enemies current position |
| enemySprite | Texture2D | Private | Enemies() | Holds the enemy sprite |
| isAlive | Boolean | Public | Enemies() | Holds the current state of the enemy |
| scoreValue | Int | public | Enemies() | Holds the score value of the enemy |
| enemyMotion | Vector2 | Public | Enemies() | holds the current motion of the enemy |
| enemySpeed | Float | Public | Enemies() | Holds how fast the enemy is moving |
| boundary | Rectangle | Public | Enemies() | Holds boundary of the enemies |
| screenBoundary | rectangle | private | Enemies() | Holds boundary of the screen to detect collision |
|  | | | | |
| playerIndex | PlayerIndex | public | PlayerIndexEventArgs() | Specifies which controller is associated with each player |
|  | | | | |
| bgTex | Texture2D | Private | ArcadeGameplay() | Holds the game background texture |
| content | ContentManager | Private | ArcadeGameplay() | Loads a new content manager for the menu content |
| enemies | Enemies | Private | ArcadeGameplay() | Loads a new enemies for handling enemy attributes |
| enemy1Sprite | Texture2D | Private | ArcadeGameplay() | Holds the enemy texture, variation 1 |
| enemy2Sprite | Texture2D | Private | ArcadeGameplay() | Holds the enemy texture, variation 2 |
| enemy3Sprite | Texture2D | Private | ArcadeGameplay() | Holds the enemy texture, variation 3 |
| enemy4Sprite | Texture2D | Private | ArcadeGameplay() | Holds the enemy texture, variation 4 |
| enemyHeight | Int | Private | ArcadeGameplay() | Holds the height of the enemies |
| enemyWidth | Int | Private | ArcadeGameplay() | Holds the width of the enemies |
| enemyLasers | Lasers | Private | ArcadeGameplay() | Creates a new lasers to handle enemy lasers |
| enemyLaserSound | SoundEffect | Private | ArcadeGameplay() | Holds the sound used for the enemies firing their laser |
| enemyLaserTex | Texture2D | Private | ArcadeGameplay() | Holds the texture for the enemies laser |
| fireDelay | TimeSpan | Private | ArcadeGameplay() | Holds how long the player must wait before being able to fire the laser again |
| gameFont | SpriteFont | Private | ArcadeGameplay() | Holds the sprite font used by game |
| gameMusic | SoundEffect | Private | ArcadeGameplay() | Holds the music used in game |
| gameMusicInstance | SoundEffectInstance | Private | ArcadeGameplay() | Holds the instance of the game music |
| playerLaserTex | Texture2D | Private | ArcadeGameplay() | Holds the texture of the player laser |
| pauseAlpha | Float | Private | ArcadeGameplay() | Holds the alpha value for pausing the screen (used for fading to black) |
| player | Player | Private | ArcadeGameplay() | Loads a new player for handling the player attributes |
| playerLasers | Lasers | Private | ArcadeGameplay() | Creates a new lasers to handle the player lasers |
| playerLaserSound | SoundEffect | Private | ArcadeGameplay() | Holds the sound used for the player firing their laser |
| explosions | SoundEffect | Private | ArcadeGameplay() | Holds the sound for player death |
| prevFireTime | TimeSpan | Private | ArcadeGameplay() | Holds the last time the player fired the laser |
| scoreValue | Int | Private | ArcadeGameplay() | Holds the score value of the enemies |
| screenBoundary | Rectangle | Private | ArcadeGameplay() | Hold the current screen bounds |
|  | | | | |
| bgTex | Texture2D | Private | EndlessGameplay() | Holds the game background texture |
| content | ContentManager | Private | EndlessGameplay() | Loads a new content manager for the menu content |
| enemies | Enemies | Private | EndlessGameplay() | Loads a new enemies for handling enemy attributes |
| bossCurrentHP | Int | Private | EndlessGameplay() | Holds the boss’s current health |
| bossLaserTex | Texture2D | Private | EndlessGameplay() | Holds the texture of the boss’ laser |
| bossPos | Vector2 | Private | EndlessGameplay() | Holds the boss current position |
| bossSprite | Texture2D | Private | EndlessGameplay() | Holds the texture of the boss |
| enemy1Sprite | Texture2D | Private | EndlessGameplay() | Holds the enemy texture, variation 1 |
| enemy2Sprite | Texture2D | Private | EndlessGameplay() | Holds the enemy texture, variation 2 |
| enemy3Sprite | Texture2D | Private | EndlessGameplay() | Holds the enemy texture, variation 3 |
| enemy4Sprite | Texture2D | Private | EndlessGameplay() | Holds the enemy texture, variation 4 |
| enemyHeight | Int | Private | EndlessGameplay() | Holds the height of the enemies |
| enemyWidth | Int | Private | EndlessGameplay() | Holds the width of the enemies |
| enemyLasers | Lasers | Private | EndlessGameplay() | Creates a new lasers to handle enemy lasers |
| enemyLaserSound | SoundEffect | Private | EndlessGameplay() | Holds the sound used for the enemies firing their laser |
| enemyLaserTex | Texture2D | Private | EndlessGameplay() | Holds the texture for the enemies laser |
| fireDelay | TimeSpan | Private | EndlessGameplay() | Holds how long the player must wait before being able to fire the laser again |
| gameFont | SpriteFont | Private | EndlessGameplay() | Holds the sprite font used by game |
| gameMusic | SoundEffect | Private | EndlessGameplay() | Holds the music used in game |
| gameMusicInstance | SoundEffectInstance | Private | EndlessGameplay() | Holds the instance of the game music |
| playerLaserTex | Texture2D | Private | EndlessGameplay() | Holds the texture of the player laser |
| pauseAlpha | Float | Private | EndlessGameplay() | Holds the alpha value for pausing the screen (used for fading to black) |
| player | Player | Private | EndlessGameplay() | Loads a new player for handling the player attributes |
| playerLasers | Lasers | Private | EndlessGameplay() | Creates a new lasers to handle the player lasers |
| playerLaserSound | SoundEffect | Private | EndlessGameplay() | Holds the sound used for the player firing their laser |
| explosions | SoundEffect | Private | EndlessGameplay() | Holds the sound for player death |
| prevFireTime | TimeSpan | Private | EndlessGameplay() | Holds the last time the player fired the laser |
| scoreValue | Int | Private | EndlessGameplay() | Holds the score value of the enemies |
| screenBoundary | Rectangle | Private | EndlessGameplay() | Hold the current screen bounds |
|  | | | | |
| bgTex | Texture2D | Private | CoopGameplay() | Holds the game background texture |
| content | ContentManager | Private | CoopGameplay() | Loads a new content manager for the menu content |
| enemies | Enemies | Private | CoopGameplay() | Loads a new enemies for handling enemy attributes |
| bossCurrentHP | Int | Private | CoopGameplay() | Holds the boss’s current health |
| bossLaserTex | Texture2D | Private | CoopGameplay() | Holds the texture of the boss’ laser |
| bossPos | Vector2 | Private | CoopGameplay() | Holds the boss current position |
| bossSprite | Texture2D | Private | CoopGameplay() | Holds the texture of the boss |
| enemy1Sprite | Texture2D | Private | CoopGameplay() | Holds the enemy texture, variation 1 |
| enemy2Sprite | Texture2D | Private | CoopGameplay() | Holds the enemy texture, variation 2 |
| enemy3Sprite | Texture2D | Private | CoopGameplay() | Holds the enemy texture, variation 3 |
| enemy4Sprite | Texture2D | Private | CoopGameplay() | Holds the enemy texture, variation 4 |
| enemyHeight | Int | Private | CoopGameplay() | Holds the height of the enemies |
| enemyWidth | Int | Private | CoopGameplay() | Holds the width of the enemies |
| enemyLasers | Lasers | Private | CoopGameplay() | Creates a new lasers to handle enemy lasers |
| enemyLaserSound | SoundEffect | Private | CoopGameplay() | Holds the sound used for the enemies firing their laser |
| enemyLaserTex | Texture2D | Private | CoopGameplay() | Holds the texture for the enemies laser |
| fireDelay | TimeSpan | Private | CoopGameplay() | Holds how long the player must wait before being able to fire the laser again |
| gameFont | SpriteFont | Private | CoopGameplay() | Holds the sprite font used by game |
| gameMusic | SoundEffect | Private | CoopGameplay() | Holds the music used in game |
| gameMusicInstance | SoundEffectInstance | Private | CoopGameplay() | Holds the instance of the game music |
| playerLaserTex | Texture2D | Private | CoopGameplay() | Holds the texture of the player laser |
| pauseAlpha | Float | Private | CoopGameplay() | Holds the alpha value for pausing the screen (used for fading to black) |
| player | Player | Private | CoopGameplay() | Loads a new player for handling the player attributes |
| player2 | Player | Private | CoopGameplay() | Loads a new player for handling the player2 attributes |
| player2Pos | Vector2 | Private | CoopGameplay() | Holds the current position of player2 |
| player2Sprite | Texture2D | Private | CoopGameplay() | Holds the texture of player2 |
| playerLasers | Lasers | Private | CoopGameplay() | Creates a new lasers to handle the player lasers |
| playerLaserSound | SoundEffect | Private | CoopGameplay() | Holds the sound used for the player firing their laser |
| explosions | SoundEffect | Private | CoopGameplay() | Holds the sound for player death |
| prevFireTime | TimeSpan | Private | CoopGameplay() | Holds the last time the player fired the laser |
| scoreValue | Int | Private | CoopGameplay() | Holds the score value of the enemies |
| screenBoundary | Rectangle | Private | CoopGameplay() | Hold the current screen bounds |
|  | | | | |
| conPlayer | PlayerIndex? | Public | GameScreen() | Holds the index of the player who triggered the events |
| isExiting | Boolean | Public | GameScreen() | Holds if a screen is currently exiting |
| isPopup | Boolean | Public | GameScreen() | Holds if a screen is a popup or not |
| otherScreenHasFocus | Boolean | Public | GameScreen() | Holds if another screen currently has focus or not |
| screenManager | ScreenManager | Public | GameScreen() | Creates a new CcreenManager to handle this screen |
| screenState | ScreenState | Public | GameScreen() | Creates a new ScreenState to handle this screen |
| transOffTime | TimeSpan | Public | GameScreen() | Holds how long it will take the screen to transition off |
| transOnTime | TimeSpan | Public | GameScreen() | Holds how long it will take the screen to transition on |
| transPos | Float | Public | GameScreen() |  |
|  | | | | |
| currentGamePadStates | GamePadState | Public readonly | InputState() | Holds the current gamepad state |
| currentKeyboardSTates | KeyboardState | Public readonly | InputState() | Holds the current keyboard state |
| gamePadWasConnected | Boolean | Public readonly | InputState() | Holds if a gamepad was connected |
| lastGamePadStates | GamePadState | Public readonly | InputState() | Holds the last gamepad state |
| lastKeyboardStates | KeyboardState | Public readonly | InputState() | Holds the last keyboard state |
| maxInputs | Int | Public const | InputState() | Holds the maximum allowed inputs |
|  | | | | |
| blankTex | Texture2D | Private | ScreenManager() | Holds the blank texture used in the fadetoblackbuffer |
| gameFont | SpriteFont | Private | ScreenManager() | Holds the game font used in game |
| input | InputState | Private | ScreenManager() | Creates a new InputState to handle this screen |
| isInitialized | Boolean | Private | ScreenManager() | Holds if the current screen is initialised or not |
| sBatch | SpriteBatch | Private | ScreenManager() | Holds the spritebatch used for drawing in game |
| screens | List | Private | ScreenManager() | Creates a new list using the gamescreen class |
| screensToUpdate | List | Private | ScreenManager() | Creates a new list using the gamescreen class |
|  | | | | |
| currentMode | Mode | Private | HighscoreScreen() | Holds the current mode option |
| dataInt | DatabaseInt | Private | HighscoreScreen() | Creates a new DatabaseInt to handle the database queries |
| gameMode | String | Private | HighscoreScreen() | Holds the currently selected game mode for queries |
| maxRows | Int | Private | HighscoreScreen() | Holds the maximum number of rows that will be draw |
| modeEntry | MenuEntry | Private | HighscoreScreen() | Creates a new menu entry to hold the mode option |
| rowDraw | Int | Private | HighscoreScreen() | Holds the number of rows in the current query |
|  | | | | |
| loadingIsSlow | Boolean | Private | Loading() | Tells the game if loading is slow, used to determine if message should be shown |
| otherScreensGone | Boolean | Private | Loading() | Tells the game is other screens have transitioned off |
| screensToLoad | GameScreen | Private | Loading() | Tells the game which screens to load |
|  | | | | |
| pos | Vector2 | Private | MenuEntry() | Holds the position of the menu entry |
| selFade | Float | Private | MenuEntry() | Holds the value for fading the selected entry |
| text | String | Private | MenuEntry() | Holds the text of the menu entry |
|  | | | | |
| content | ContentManager | Private | MenuScreen() | Loads a new content manager for menu content |
| menuEntries | List | Private | MenuScreen() | Creates menu entries as a new list |
| menuMusic | SoundEffect | Private | MenuScreen() | Holds the music for the menu |
| menuMusicInstance | SoundEffectInstance | Private | MenuScreen() | Holds the music for the menu |
| MenuTitle | String | Private | MenuScreen() | Holds the menu title |
| selEntry | Int | Private | MenuScreen() | Holds the integer of entry selection |
|  | | | | |
| message | String | Private | MsgboxScreen() | Holds the message to be displayed |
| gradientTexture | Texture2D | Private | MsgboxScreen() | Holds the texture of the gradient used in the background |
|  | | | | |
| currentColour | laserColour | Public static | OptionsScreen() | Initial menu option for laserColour |
| currentMusic | music | Public static | OptionsScreen() | Initial menu option for music |
| currentShip | playerShip | Public static | OptionsScreen() | Initial menu option for playerShip |
| laserColourentry | MenuEntry | Public | OptionsScreen() | Initialises menu entries as menu entry class |
| musicEntry | MenuEntry | Public | OptionsScreen() | Initialises menu entries as menu entry class |
| playerShipEntry | MenuEntry | Public | OptionsScreen() | Initialises menu entries as menu entry class |
|  | | | | |
| connection | OleDBConnection | Private | DatabaseInt() | Initialises a new connection |
| conString | String | Private | DatabaseInt() | Tells the game where to look for the database |
| dataRowC | DataRowcollection | Private | DatabaseInt() | Initialises the datarow collection |
| dataset | DataSet | Private | DatabaseInt() | Initialises the dataset |
| playerID | Int | Private | DatabaseInt() | Holds the playerID |
| rowDraw | Int | Private | DatabaseInt() | Tells the game how many rows to display |
| transaction | OleDBTransaction | private | DatabaseInt() | Holds the initial transaction value |
|  | | | | |
| graphics | GraphicsDeviceManager | Private | Game() | Creates a new instance of GraphicsDeviceManager to handle graphics devices |
| preloadAssets | String | Private | Game() | Preloads assets to reduce strain |
| sBatch | SpriteBatch | Private | Game() | Holds the spritebatch used for drawing in game |
| screenManager | ScreenManager | Static readonly | Game() | Creates a new screenManger for handling this screen |

### Full Code