|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Description: https://sharepoint.midkent.ac.uk/sites/marketing/MidKent%20Logo%20blue%20email.jpg  **Assignment front sheet** | | | | | | | |
| Learner name | | | | Assessor name | | | |
| Christopher Edwicker | | | | **Karen Scott** | | | |
| Date issued | | | Deadline | | Submitted on | | |
| **21st April 2014** | | | **2nd May 2014** | | 1st May 2014 | | |
| Qualification | | Unit number and title | | | | | |
| **BTEC Level 3 Extended Diploma in IT** | | **Unit 22 Developing Computer Games (Y/601/7660)** | | | | | |
| Assignment Title | | **U22\_Asst2\_P3\_P7(part)\_M1\_D2\_Design\_a\_Game** | | | | | |
| In this assessment you will have opportunities to provide evidence against the following criteria.  Indicate the page numbers where the evidence can be found. | | | | | | | |
| Criteria reference | To achieve the criteria the evidence must show that  the student is able to: | | | | | Task no. | Location of Evidence |
| P3 | produce a design for a computer game for a given specification | | | | | 1 |  |
| P7 (part) | produce technical documentation for a (design for a) computer game | | | | | 1 |  |
| M1 | determine appropriate data types for a computer game and show how they are declared | | | | | 1 |  |
| D2 | explain how the structure and design of a game can assist in  maintenance and capacity for extension. | | | | | 1 |  |
| Learner declaration | | | | | | | |
| I certify that the work submitted for this assignment is my own and research sources are fully acknowledged.  Learner signature: Date: | | | | | | | |
|  | | | |  | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Assessor's comments** | | | | | | | |
| Qualification | | **BTEC Level 3 Extended Diploma in IT** | | Assessor name | **Karen Scott** | | |
| Unit number and title | | **Unit 22 Developing Computer Games (Y/601/7660)** | | Learner name |  | | |
| Assignment title | | **U22\_Asst2\_P3\_P7(part)\_M1\_D2\_Design\_a\_Game** | | | | | |
|  | | | | | | | |
| Grading criteria | Achieved? | | Comments | | | |
| P3 |  | |  | | | |
| P7(part) |  | |  | | | |
| M1 |  | |  | | | |
| D2 |  | |  | | | |
| General comments | | | | | | | |
|  | | | | | | | |
| Action plan | | | | | | | |
|  | | | | | | | |
| Assessor signature | |  | | | Date |  | |

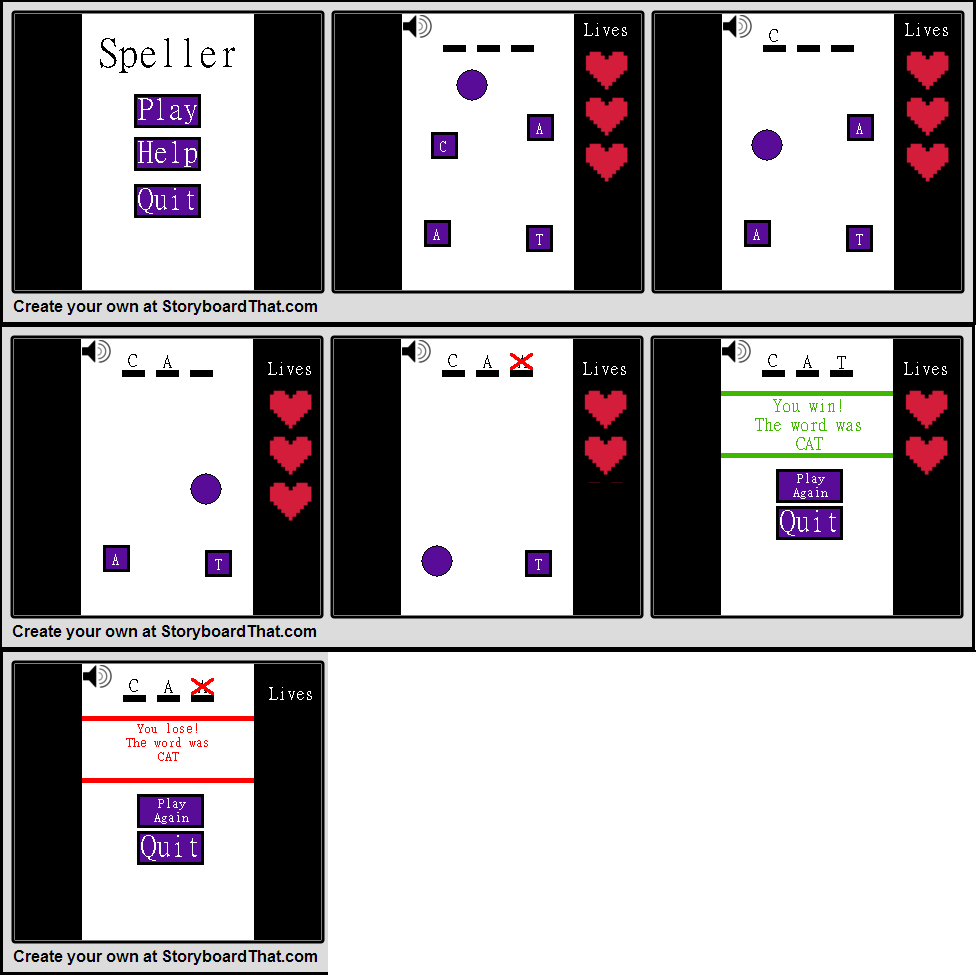
**Speller**

In this assignment I will be designing a game to be used on the Medway dyslexia house website.

My game, in keeping with the theme of being used for a dyslexia website is a spelling game in which, the player will hear a word being said, the game will then begin to slide letters towards the players character. The player must dodge incorrect letters and hit the correct letters in the correct order to win the game. If an incorrect letter is hit at the wrong time, the player will lose a life. When the game ends, the game will ask if the player wants to play again, if the player does, a new word will be chosen and he game will start again.

**Storyboard**

First, we have a storyboard of how I expect the game to play out from start to finish. Below the storyboard, I will explain each pane.



|  |  |
| --- | --- |
| Pane 1 | Here we see the main menu, with three different buttons. The Play button will begin the game, the Help button will show the help screen and the Quit button will quit the game. |
| Pane 2 | In this pane we see the game as it is at the start. Empty spaces for the letters the player will collect, a speaker icon for the player to click if they need to hear the word again, the character which can be controlled with A and D to move left and right respectively, letters that the player can collect an finally the life counter on the right. |
| Pane 3 | We can see that as the player collides with a letter, the game checks to see if he letter was first in the order. If it was then the letter is added to the top display. |
| Pane 4 | The player then moves to the next letter, and the same process is followed. |
| Pane 5 | Here the player collected the wrong letter, the letter that was collected appears, a red X appears over it and then the letter and cross are removed as well as one heart on the right. |
| Pane 6 | Once the final letter is collected, the win screen is displayed. |
| Pane 7 | If the player runs out of lives and has not collected all of the letters in the correct order, this pane will be displayed in place of pane 6. |

**Class diagram**

Below is a full class diagram for each world and actor I plan to use. These may change later on in development if I find something is missing.

**Worlds**:

|  |  |
| --- | --- |
| Name | Menu |
| Attributes | Width – 600px  Height – 800px  (obj\_Border is 100px so playable world is 400, 800)  Background – crumpled-paper.jpg |
| Methods | prepBackground() – Prepares the background, adds obj\_Border to the left and right and the word “Speller” in top centre.  prepButtons() – Create obj\_btnPlay, obj\_btnHelp, obj\_btnQuit. |

|  |  |
| --- | --- |
| Name | GameWorld |
| Attributes | Width – 600px  Height – 800px  (obj\_Border is 100px so playable world is 400, 800)  Background – crumpled-paper.jpg |
| Variables | playerLives – Holds the players number of lives, default 3.  wordToSpell – The word the player must spell.  currentLetters – The players current letters.  playerWin – Has the player won? Default false. |
| Methods | startGame() – Triggered by obj\_btnPlay, triggers all methods below.  genWord() – Generate the wordToSpell. |

Note code:  
if(!playerWin && playerLives == 0){  
addObject(x, y, obj\_winScreen)  
} else if{  
addObject(x, y, obj\_loseScreen)  
}

|  |  |
| --- | --- |
| Name | GameWorld |
| Attributes | Width – 600px  Height – 800px  (obj\_Border is 100px so playable world is 400, 800)  Background – crumpled-paper.jpg |
| Variables | None. |
| Methods | loadHelp() – Loads help text.  genLetters() – Generates a letter box at random. |

Actors:

|  |  |
| --- | --- |
| Name | Obj\_Border |
| Attributes | Width – 100px  Height – 800px  Image – obj\_Border.jpg |
| Variables | None. |
| Methods | None. |

|  |  |
| --- | --- |
| Name | Obj\_btnPlay |
| Attributes | Width – px  Height – px  Image – obj\_btnPlay.jpg |
| Variables | None. |
| Methods | changeWorld() – Changes to GameWorld  Triggers startGame() in GameWorld. |

|  |  |
| --- | --- |
| Name | Obj\_btnHelp |
| Attributes | Width – px  Height – px  Image – obj\_btnHelp.jpg |
| Variables | None. |
| Methods | showHelp() – Changes world to HelpWorld. |

|  |  |
| --- | --- |
| Name | Obj\_btnQuit |
| Attributes | Width – px  Height – px  Image – obj\_btnQuit.jpg |
| Variables | None. |
| Methods | quit() – Closes the game. |

|  |  |
| --- | --- |
| Name | Obj\_btnPlayAgain |
| Attributes | Width – px  Height – px  Image – obj\_btnPlay.jpg |
| Variables | None. |
| Methods | changeWorld() – Changes to GameWorld  Triggers startGame() in GameWorld. |

|  |  |
| --- | --- |
| Name | Obj\_character |
| Attributes | Width – px  Height – px  Image – obj\_character.jpg |
| Variables | hoizSpeed – Players horizontal speed. |
| Methods | movePlayerHoriz() – Moves player left or right depending on key held at horizSpeed. Stops player from going out of bounds. |

|  |  |
| --- | --- |
| Name | Obj\_letterBox |
| Attributes | Width – px  Height – px  Image – obj\_character.jpg |
| Variables | vertSpeed – objects vertical speed.  myLetter – The objects letter.  myNumber – The objects number in the sequence. |
| Methods | moveObjVert() – Move the player up at vertSpeed without stopping.  collision() – On collision remove instance of object.  genLetter() – Generate a random letter and put it on the box and assign the letter a number based on the letters position in the word. |

|  |  |
| --- | --- |
| Name | Obj\_winScreen |
| Attributes | Width – px  Height – px  Image – obj\_winScreen.jpg |
| Variables | wordToSpell – The word for the player to spell. |
| Methods | showWord() – Place wordToSpell in correct place on the object. |

|  |  |
| --- | --- |
| Name | Obj\_loseScreen |
| Attributes | Width – px  Height – px  Image – obj\_loseScreen.jpg |
| Variables | wordToSpell – The word for the player to spell. |
| Methods | showWord() – Place wordToSpell in correct place on the object. |

|  |  |
| --- | --- |
| Name | Obj\_speaker |
| Attributes | Width – px  Height – px  Image – obj\_speaker.jpg |
| Variables |  |
| Methods | playWord() – Plays the word the user needs to spell. |

The code note half way through is a note for myself on how the game will determine the winner.

**Data dictionary**

Next, we have the data dictionary listing all of the objects and all of the variables I will be using, again, this may change as the game is developed.

Speller Game data dictionary

Objects:

* obj\_Border - None
* obj\_btnPlay - None
* obj\_btnHelp - None
* obj\_btnQuit - None
* obj\_btnPlayAgain - None
* obj\_character – horizSpeed.
* obj\_letterBox – vertSpeed, myLetter, myNumber.
* obj\_winScreen – wordToSpell.
* obj\_loseScreen – wordToSpell.

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Type | Validation | Purpose |
| horizSpeed | Int | TBD | Determines the horizontal movement speed of the character. |
| vertSpeed | Int | TBD | Determines the vertical speed of the letterbox objects. |
| myLetter | String | A-Z | Holds the letter the object is given. |
| myNumber | Int | TBD | Holds he position in the number the letter equates to. |
| wordToSpell | String | TBD | Holds the final word the player must spell. |
| playerLives | Int | 0-3 | How many lives the player currently has. |
| currentLetters | Int | TBD | Keeps track using myNumber of what letters the character has collected. |
| playerWin | Boolean | TBD | Determines whether the player has won or not and displays the correct screen depending on the outcome. |

**Flow chart**

Finally, below is a flow chart showing the process the character object will go through to decide whether or not the player has the correct letter as well as if the player has won or not.



**EVENT-ACTION TABLE**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **SoftwareTitle:** | | | Speller | | **Design date:** | 01/05/2014 | |
| **Object** | **Event** | | | **Action** | | | **Test Data and expected result** |
| Obj\_border | None. | | | None. | | | None. |
| Obj\_btnPlay | changeWorld()  startGame() | | | Move to the game world.  Starts the game. | | | Switch to the game world.  The game begins. |
| Obj\_Help | showHelp() | | | Changes to HelpWorld | | | Switches to the HelpWorld |
| Obj\_btnQuit | Quit() | | | Closes the game | | | Game closes |
| Obj\_btnPlayAgain | changeWorld()  startGame() | | | Move to the game world.  Starts the game. | | | Switch to the game world.  The game begins. |
| Obj\_character | movePlayerHoriz() | | | Moves player left or right depending on key held at horizSpeed. Stops player from going out of bounds. | | | Player moves left and right using A and D, player cannot go out of bounds. |
| Obj\_letterBox | moveObjVert()  collision()  genLetter() | | | Move the player up at vertSpeed without stopping.  On collision remove instance of object.  Generate a random letter and put it on the box and assign the letter a number based on the letters position in the word. | | | Player moves vertically.  Instance is removed.  Random letter generated and placed on the sprite. |
| Obj\_winScreen | showWord() | | | Place wordToSpell in correct place on the object. | | | wordToSpell is in the correct place |
| Obj\_loseScreen | showWord() | | | Place wordToSpell in correct place on the object. | | | wordToSpell is in the correct place |
| Obj\_Speaker | playWord() | | | Plays the word the user needs to spell. | | | Play audio file that the user needs to spell. |
| **Design prepared by:** | | Chris Edwicker | | | **Design date:** | 01/05/2014 | |

**Maintenance and Expansion**

The way I intend to develop this game will allow for the capacity of expansion in the way of adding new words that can be used in the game. I will do this by using an array to store the words, each word will have a separate number in that array. The letter generator will choose a random number and use the word that correlates to that number.

This allows for easy expansion as all that needs to be done to add new words to the game is add them to the array and using functions such as “array.length” when defining parameters for the word choosing, it will always choose a word and will never choose a word that is not associated with a word.

I have also designed the game to use variables that clearly explain what they do, and also by creating a data dictionary to give a description of what each variable does, the maintenance for this game will be very easy should it ever need changing. Should that ever be the case, any developer should easily be able to look at the code and see what everything does easily, and if they get stuck, the data dictionary should help clear up any queries.