3D Modelling and Animation (F21MA)

Flex Project
Professor Mike Chantler

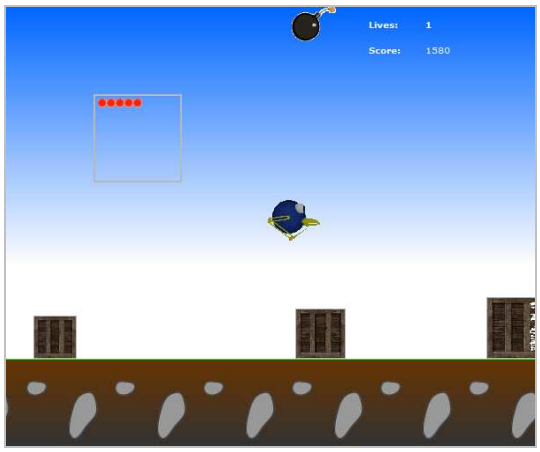
Drew Forster
062446228
Ulysse Vaussy
085167164

3D Modelling and Animation (F21MA)
Flex Project
Professor Mike Chantler

Part 1: Group Report

Executive Summary

In their very first game; two robots, Bip and Bobot, are in a competition to run as far as possible. A surprise is in store for them however; as a local miscreant has placed boxes all over the robots favourite running spot. It's a well known fact that boxes are, in fact, a robot's worst enemy. Bip and Bobot are hard-wired to withstand a number of knocks but after this they will require reprogramming. It is your mission to avoid as many boxes as possible as they race along a grassy field – or if you think you can control them, a dungeon.



Bip leaps over a box with only one life left

At your disposal you will have the ability to throw balls at objects at the balls but this comes at a cost:

- You only have a limited number of balls and should you be lucky enough to encounter a power-up you need to throw a ball to gain it!
- The balls are energetic and will bounce around the screen until they hit something – including you!

Game Objectives and Play

Objective

The main objective is to continue the game as long as possible. The challenge is that characters will lose a life each time they collide with a box or ball. The player also has a set of balls which they may throw towards targets; including special power-ups which occasionally appear.

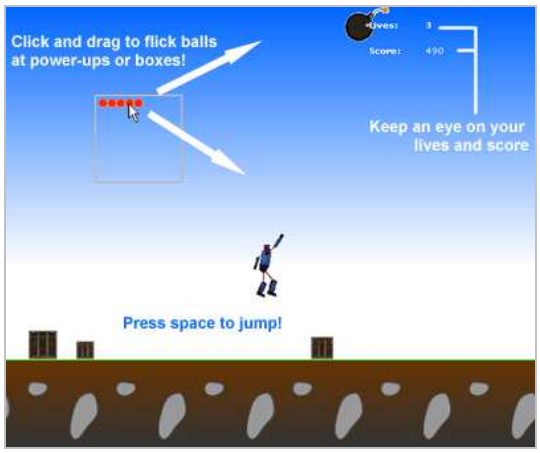


Image from the help screen within the game featuring Bobot

Controls

The robots run automatically allowing players to concentrate on jumping (via the space bar) and throwing the balls (through clicking and dragging the mouse whilst the balls are within the box). These simplistic controls reduce the learning curve and create a game which is fun and addictive from the start.

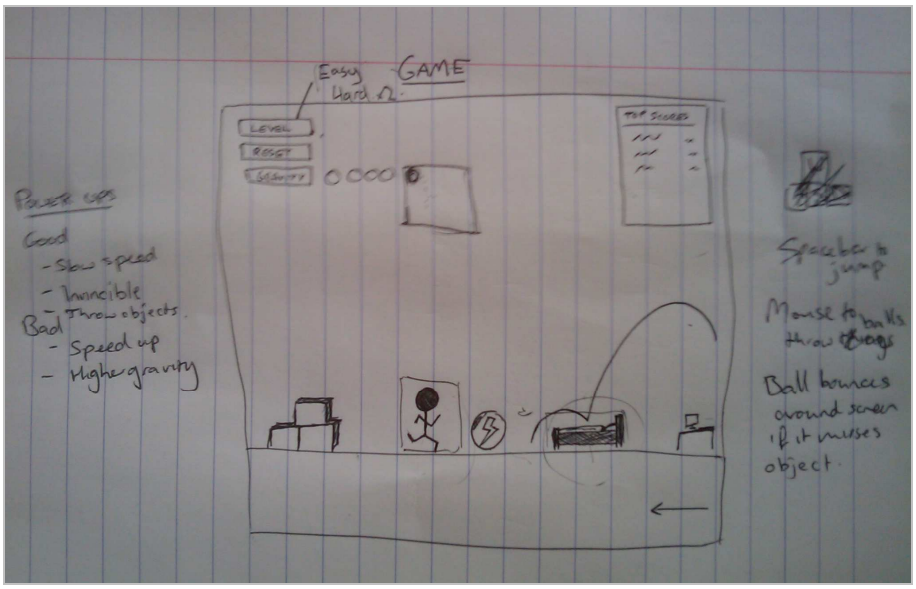
Scoring

The players score relates to how far they have travelled. A multiplier is applied based on the difficulty selected. The multiplier must make a choice: Play for less on a harder difficulty for a lot of points or play for longer on an easier level. Players can also choose to submit scores into the global score board; creating a competitive but fun atmosphere.

Game Development

The game was developed iteratively using a series of prototypes. This reduced the risk of creating misinformed requirements by constantly evaluating and making decisions regarding the development.

Initial Game Design



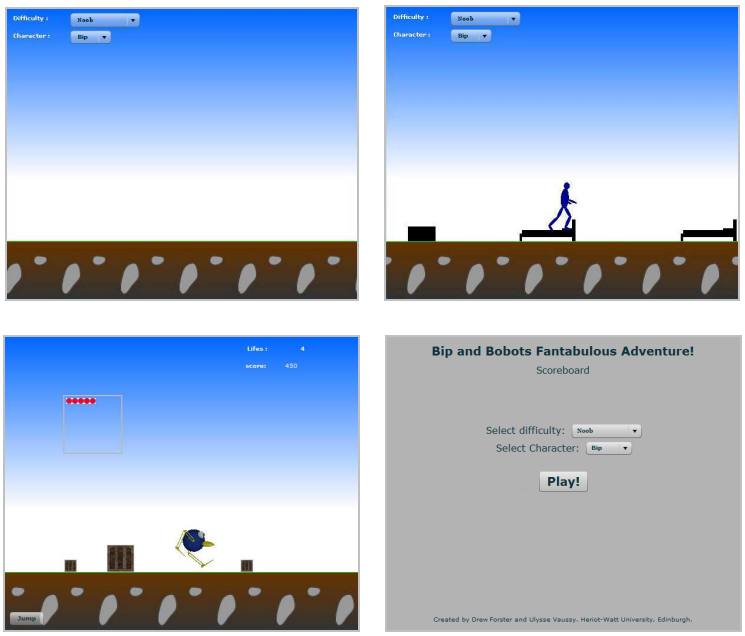
Hand drawn sketch of the initial game design

An initial design of the game was sketched early in development. This was performed in a relaxed environment allowing ideas to be included and visually represented. The ideas showing within this sketch are noted in the table below:

Implemented in final game	Not Implemented in final game
Moving background with stationary character	Different obstacle shapes with different properties (for example: A bouncy bed)
Balls to throw at objects (which continues bouncing around if it misses)	Power-ups which negatively affect the player's character
Power-ups	
A global scoreboard	
Multiple levels of difficulty	
Spacebar to jump and mouse to throw balls	

Table showing the implementation resulting from the initial sketch

Prototypes



Development of functionality through the prototypes one to four (top left to bottom right)

The images above show the progression of the game development through the prototyping stages; the prototype requirements and conclusions are displayed in detail below:

Prototype One

Create an application with a moving background using the same image multiple times. The background speed should be modifiable through a combo box.

Conclusions: The development of code to move smoothly across the screen created a base for objects and characters.

Prototype Two

Evolve prototype one to include image based characters and obstacles; the obstacle images must be different shapes and not overlap. The character and obstacle should move at the same speed as the background image. The character code and images are to be taken from Professor Mike Chantler's examples.

Conclusions: The prototype showed the complexity required to create smooth movement and character behaviours. The initial concept of different shapes with different properties was dropped at this stage to focus on development of other functionality.

Prototype Three

Create a character walk cycle from Autodesk 3ds Max and import it into the animation. The prototype should include collision detection between character and obstacles. A simple jump animation should also be included for the character – accessible by a button. The initial ball design was also included; this allowed balls to be thrown around the screen though no collisions took place.

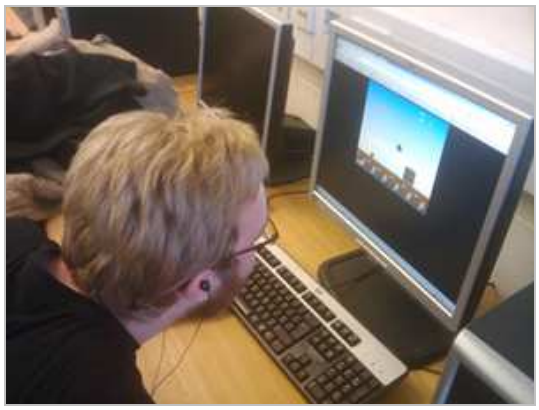
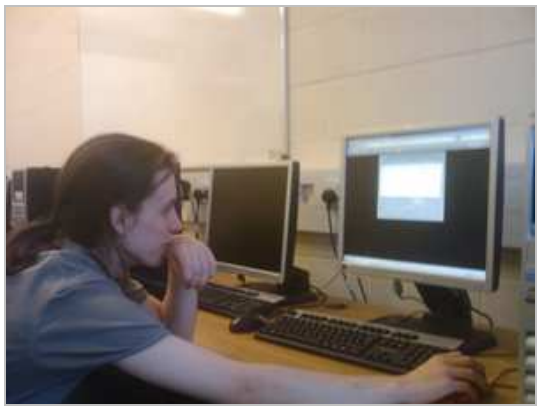
Conclusions: Simplifying the game to boxes created a simple, addictive gameplay structure. The character animation fit in well with the game but a fall animation was required.

Prototype Four (Beta release)

Prototype four is split into two main functional parts: implementation of additional screens and refining of the existing game functions. A title screen, end screen and help screen are to be created to navigate around the game. The database and PHP scripts to access it are also to be created. The game should be refined so that collision detection is included for the balls, additional animations are added and the space bar is mapped to the jump function.

Conclusions: The main functionality the game set out to achieve was realised. This created the opportunity for additional functionality based on user perceptions of the game.

User Testing

Photos taken at the user testing of the system

User Testing Conclusions: Users noted that the gameplay in the easy mode was perhaps too slow. It was also suggested to include more variation in the gameplay to maintain user involvement. Users commented on the good use of a single key and mouse functionality allowing them to access both sets of functionality at the same time easily.

Prototype Five (Final release)

The final system will introduce different power-ups to enhance gameplay; these will be accessed by the ball modules already present within the game. The speed issue within the gameplay will be corrected to create a more user friendly game. The scoreboard will be implemented and displayed within the main screen allowing players to compete with each other.

Conclusions: The final release within the scope of this project showed a simple game made addictive by well developed animations, complex and accessible gameplay and a competitive environment.

Overall Design

Modular Diagram

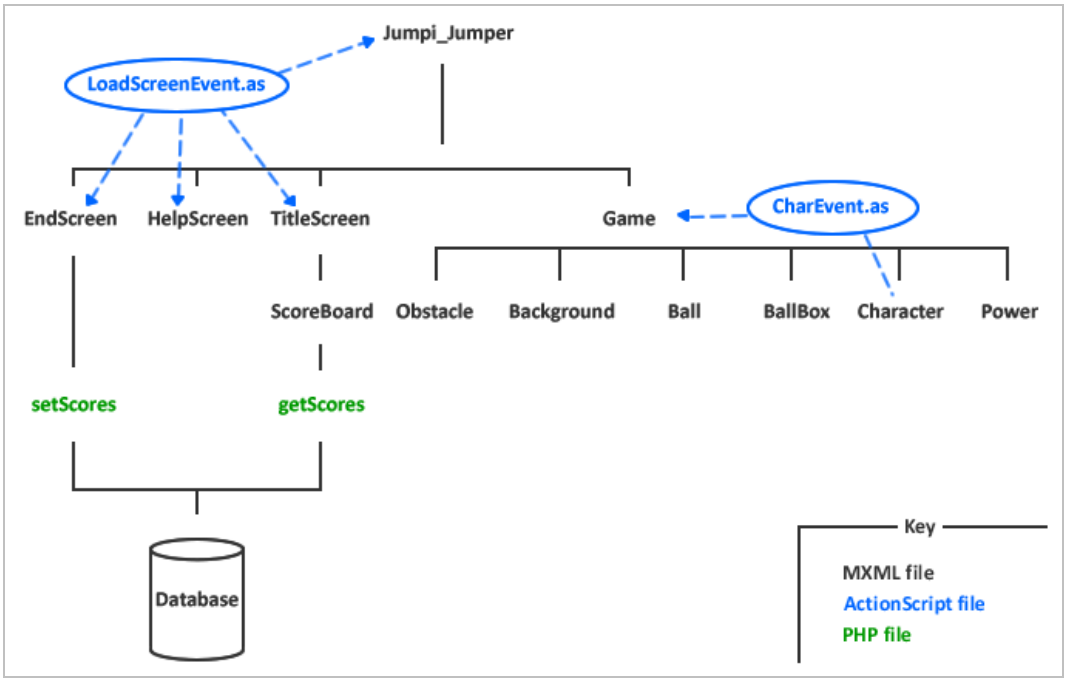


Diagram showing modular structure of game

The modular diagram above shows the links between modules within the final game. There are four main types: the database, PHP scripts, MXML files and ActionScript classes.

Database

The database is used to store the names, score and dates of scores entered through the system. The setScores PHP script will insert the data and the getScores PHP script will retrieve all the data currently in the database.

PHP

setScores – *The setScores file takes a username and score and inserts them into the database with the current date. It is the responsibility of the EndScreen MXML module to call this file. If successful, the scoreID is returned. This is passed through to the TitleScreen module then the ScoreBoard module. It is used in the ScoreBoard to highlight the recent score, allowing the user to easily locate where they are in the leader board.*

getScores – *The getScores returns all data currently held within the database as an XML output. The ScoreBoard MXML component is responsible for calling the PHP script to retrieve the data. The data is then used to populate the ScoreBoard.*

Note: All PHP files have substantial error catching and return useful error messages to inform users of an errors which have occurred.

MXML Files

Background – *The BackGround component is responsible for moving the background images across the screen creating the effect of motion within the game. The BackGround component is held within the Game component, which is responsible for creating and updating the BackGround module.*

Ball – *The Ball module is used to create and manipulate the balls used to throw at objects and power-ups. The Ball module is part of the Game module and can interact with the BallBox, Character, Power and Obstacle modules through collision detection functions within the Game module.*

BallBox – *The BallBox module is used as an area to store Ball components and is the only place where players can control Ball components. When outside the box the Ball components move with the momentum they were issued rebounding off walls until they hit an object. The BallBox module is held within the Game module and interacts with the Ball modules.*

Character – *The Character module contains the functionality used to show the animated character (seemingly) traversing the level, jumping and falling. The Character module is used within the Game module and interacts with the Ball and Obstacle components through the collision detection functions. When the Character is set to invincible a CharEvent is used to inform the Game module.*

EndScreen – *The EndScreen module is shown to the user when the game is finished, it displays their score and prompts them to enter a name to submit their score or to play again. Whichever choice the user picks, the TitleScreen is loaded by firing a LoadScreenEvent. If the*

user has chosen to submit a score, it is submitted through the setScores PHP module and a scoreID is passed through to the TitleScreen.

Game – *The Game module contains most of the code which accesses other components. It is responsible for creating and controlling Character, Obstacle, Power, Ball and BallBox components. It also contains the collision detection functions for all components present within it to check if a collision occurs and the action to take to the responsible modules.*

HelpScreen – *The HelpScreen module contains details about the game. This includes an image showing how to play as well as a short description of the objective. The HelpScreen can be loaded from the TitleScreen through a LoadScreenEvent and vice versa.*

Obstacle – *The Obstacle module is used to create and manipulate the obstacles which pass along the screen. The Obstacle module is held within the Game module, which is responsible for controlling it. It can interact with the Character and Ball modules through the collision detection within the Game module.*

Jumpi_Jumper – *The Jumpi_Jumper module is the main class which runs the application. It is used to handle all the screens within the system. It interacts with the EndScreen, Game, HelpScreen and TitleScreen by listening for a LoadScreenEvent.*

Power – *The Power module is used to create and control power-ups which pass along the top of the screen. Each power-up alters the gameplay in a slightly different way so the Game module retrieves the type if it is struck by a Ball module.*

ScoreBoard – *The ScoreBoard module is used to display the scores currently recorded within the database. It retrieves the data through the getScores PHP module and is placed within the TitleScreen module. If a scoreID is supplied, the ScoreBoard will highlight and move to the score to show the player where they are positioned in the table.*

TitleScreen – *The TitleScreen module is the first screen the player will see. It is used to display the current high scores, allow the users to select a difficulty, allow the users to select a character, preview the character selected, play the game or view the HelpScreen module.*

Either the Game or HelpScreen module is loaded through a LoadScreenEvent which is handled within the Jumpi_Jumper module.

ActionScript Classes

CharEvent – *The CharEvent is used to inform the Game module that a Character is temporarily invincible (after the character hits a box)*

LoadScreenEvent – *A LoadScreenEvent is used whenever a new screen is required on the Jumpi_Jumper module.*

Joint Conclusions

Technology Evaluation

As the project revolved around Adobe Flex and AutoDesk 3ds Max it was necessary to learn two new pieces of development software for both members of the group. This created a large stress on the project initially which pushed it slightly behind schedule. This is to be expected with any new technology; however, taking on 2 new environments increases the risk.

Creating models in 3ds Max was a good learning experience. Visualisation of different cameras, angles, textures and lighting creates an instant response atmosphere; this reduces the time required to plan key elements but does increase the need for trial and error. It should also be noted that the 3d output is very impressive for developers with no previous experience in such a powerful environment.

As a means for creating simple animated games; Adobe Flex seems a little out of its depth. Whilst it offers a coding environment to create games it sometimes seems too heavy to handle even slightly complex tasks – especially including images. It should be noted of course that Adobe Flex was developed for use in e-commerce originally; in these less demanding atmospheres it would certainly be fast to develop sleek prototypes.

Game Evaluation

The final release of the game was to a high standard with a lot of reliable functionality. In the user testing it became apparent that the users enjoyed playing and would happily compete for high scores within the game.

The game was developed to be simple but reliable enough to become addictive. The final game has retained these goals and added some key functionality in the form of power-ups and balls to throw at objects; creating additional aspects of gameplay and when mastered do allow higher scores. Overloading the game with more functionality could take away the easy to learn and "pick up and play" aspect at the heart of the game.

Instead of adding additional functionality it was reasoned that implementing the existing functionality would be the best way to extend the game. One suggested idea was a story mode which would contain numerous levels with increasing difficulty and a storyline to captivate the player. Additionally, it could be possible to gain further animations (for example shooting) by progressing through the game offering a reward system.

Project Evaluation

The project was developed iteratively through prototypes, this allowed the requirements to be regularly reviewed and updated. This meant it was possible to learn from mistakes, experiences and take opportunities that the developers found through creating initial prototypes. As an example; it was quickly realised that the background could be used as a foundation for other moving objects, allowing for more time to be spent later in the program adding additional functionality.

Splitting the code into modules allowed individual programmers to be working on the same game at the same time; problems were only met because of the requirement to have one "main" class which would handle most of the game components. Working to deadlines also allowed key modules to be produced for prototypes; this resulted in a program which was consistently built up and, as a result, at the deadline was complete and contained additional functionality.

3D Modelling and Animation (F21MA)
Flex Project
Professor Mike Chantler

Part 2: Individual Report

Ulysse Vaussy
085167164

Summary of Role

My participation to this project was concerning the design of the game as well as the coding.

For the design I participate to find the basic principle (robot jumping over obstacles)

And to figure how to implement specific requirement, for example: second order motion and dynamic objects.

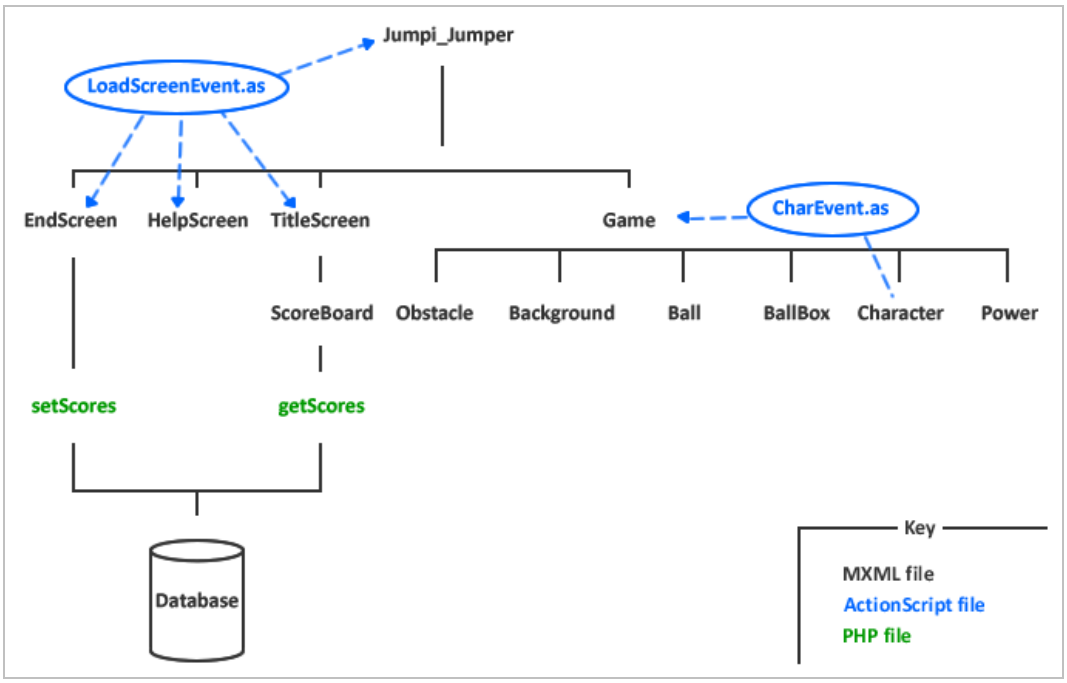


Diagram showing modular structure of game

My participation for the coding was concerning the fallowing module:

Ball

BallBox

Character

Power

Game

Balls

Parts of "init()" related to the module I code

Score

Ulysse Vaussy

Signed: ……

Drew Forster

Signed: ……

Module Descriptions

Ball

This module is use to draw and manipulate balls that will be through on Object, that Include dragging and throwing ball.

This module is use in the game module to interact with, BallBox, Power Obstacle and Character.

30% of the code is from Ulysse Vaussy
The calculation of the new position of the ball and the bouncing comes from Mike Chantler example.

Through the ball
To adapt this module for Jumpi_Jumper it was necessary to have different way to drop the ball. If the player drop the ball inside of the Ballbox he is still able to catch it again, but if the ball is dropped when going out of the boxes the event listener on mouse down is deleted.

The issue that has been solve, was concerning the calculation of the velocity when throwing the ball. In the example the value of oldX and oldY was set in the event onEnterFrame of the module Ball, but include in the game those value has to be set in the event onEnterFrame of the main game, otherwise oldX and oldY was always equal to x and y when performing the calculation.

```
init(e:Event)
```

Initialisation of event listener when the module is create

`initBall()`

Initialisation of event listener call when ball are reset

`onEnterFrame(e:Event)`

```
Calculate new position of ball for every new frame
```

`mouseDownHandler(e:MouseEvent)`

```
Start to drag the ball when mouse button down
```

`mouseUpHandler(eMouseEvent)`

```
Sop dragging the ball when mouse button realised
```

`StopDraging()`

```
Sop dragging the ball when called (when ball is out of BallBox)
The mouseDownHandler is deleted
```

`destroy()`

```
Set ball invisible and it position at x=0 y=0.
```

`getDraging()`

```
Return true if ball currently dragged
```

BallBox

This module is an empty module to define the size and the border of the box that contain balls.

This module is used in the Game module to interact with balls

100% of the code is from Ulysse Vaussy

Character

This module is use to draw and manipulate the character.
It contains the number of lives velocity, gravity and it uses an array of image to animate the character when it is walking jumping and falling.

This module is used in the Game module to interact with every object of the environment and keyboard inputs.

80% of the code is from Ulysse Vaussy
The principle of repeater using an array of image to simulate the animation of the character was taken from Mike Chantler example.

`setCharacter(character:String)`

Set the variables for the specific character

`initAnimation()`

Initialise the list of image and set every image invisible

`onNextFrame(e:Event)`

Select the appropriate image to display for each new frame depending on the Character and the current action (walking, jumping or falling)

Change image every 3 frames

```
walk()
```

Make the character walking, activate enter frame event and set start image and length of animation

```
jump()
```

Make the character jumping, set start image and length of animation

```
fall()
```

Make the character falling, set start image and length of animation

```
hit()
   Call when character is hit by something.
   Decrease lives by 1 and set character invincible
```

```
setGravity(level:Number)
```

Set the value of gravity depending on the level of difficulty

```
increaseyVelocity()
```

Increase the height of the jump of 1 up to 14.

Call when the battery power is hit

```
stopJumping()
```

Set jumping = false

`isJumping()`

Return jumping

`startFalling()`

Set falling true

`getLifes()`

`Return lives`

`setLives(number:int)`

Set live to number

Power

This module is use to draw and move powers. Powers appears every 1000 frames and has to be catch by throwing ball on it. They appears on the top right hand side of the screen and move to the left.

This module is used in the module game to interact with balls and activate the power

100% of the code is right by Ulysse Vaussy.

`onNextFrame(e:Event)`

Move the image to the left and destroy it if it go out of the screen

`newPower(type:Number)`

Initialise a new power with it Type number

Set the appropriate image visible

`Destroy()`

Set module invisible and delete the event listener (stop the movement)

`setSpeed(newSpeed:int)`

Set speed of the of the power

`getType()`

Return type, [0 live, 1 higher jump, 2 new balls, 3 Bombe]

`isDestroyed()`

Return true if the power is destroyed

The creations of a newPower initialise “Type” that represents the type of power and set the appropriate image visible and other hidden.

The method “Destroy” set destroyed at true and hide the all model.

Setter and Getter are used to set the speed, get the type of power and check if the power is destroyed.

Class CharEvent

This module is used to make bubbling working so events are so data are pass back from character.

The invincible event is declared in this class to have an accessible event type.

10% of the code is right by Ulysse Vaussy.
The bubble function comes from Mike Chantler example.

This module is used in character and game module in order to make the event going from the character up to the game module.

Game

This is the main module; therefore each of us had to implement his on initialisation and functions in it. My part was to set my initialisation and develop the calculation of score and lives as well as the collision detection.

90% of the code concerning Ulysse implementation is right by Ulysse Vaussy.
The bouncing ball come frome Mike Chantler example and the collision detection is inspired from examples.

In the init function all elements are set at there initial position and value.

The onEnterFrame function is the equivalent of the main loop of the game.

Every 10 frame the score is incremented depending on the speed of the game, and

The collision detection for balls and obstacle are call.

`handleKeyDown(event:KeyboardEvent)`

Keyboard listener fro jumping

`characterInvincible(e:CharEvent)`

Set the character invincible for a number of frame

`initBalls()`

Set balls positions

`BallColision()`

Make the ball bouncing against sides of the screen.

Detect collision of balls with other element, Power, Character, Obstacles

And perform the appropriate action. If the ball hit.

Finish the dragging if the ball is out of BallBox

In order to make the collision detection more realistic we don't use the size of the image

(that is bigger than the robot) but parameters that are closer to the real size of the robot.

What I have gained from this project

From this project I discovered flex and gained a good understanding of it. It was also an interesting experience of using techniques and tools for game development, such as collision detection, and movement.

The team working was a very positive with a good exchange of skills.

It was an interesting project, and the part of creativity was a good thing to get involve personally in the project.

3D Modelling and Animation (F21MA)
Flex Project
Professor Mike Chantler

Part 2: Individual Report

Drew Forster
062446228

Summary of Role

Overview

For the project I created the background module and obstacle module before focusing on the database, screen development and layout of the program.

Detailed

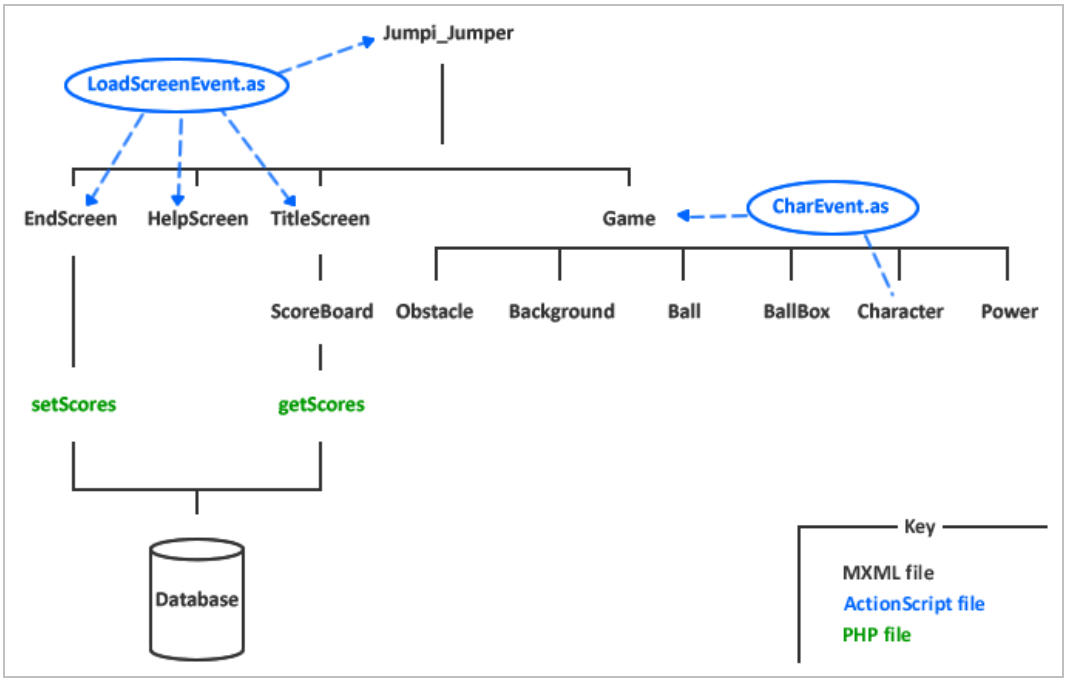


Diagram showing modular structure of game

Jumpi_Jumper **(student percentage: 100% | course percentage: 0%)**

Game **(student percentage: 90%| course percentage: 10%)**

Background **(student percentage: 95% | course percentage: 5%)**

Obstacle **(student percentage: 95% | course percentage: 5%)**

EndScreen **(student percentage: 95% | course percentage: 5%)**

HelpScreen **(student percentage: 80% | course percentage: 20%)**

TitleScreen **(student percentage: 90% | course percentage: 10%)**

ScoreBoard **(student percentage: 100% | course percentage: 0%)**

setScores.php **(student percentage: 100% | course percentage: 0%)**

getScores.php **(student percentage: 100% | course percentage: 0%)**

Database **(student percentage: 100%| course percentage: 0%)**

Ulysse Vaussy

Signed: ……

Drew Forster

Signed: ……

Module Descriptions

Jumpi_Jumper

This module is used to handle all screen control within the system. This method simply loads a canvas into the main application depending on what screen string is sent in the LoadScreenEvent. It simplifies the layout of the system and ensures that there is only ever one canvas on screen (reducing the resource usage).

Background

This module controls the background image within the game. The background moves at a set speed to create the illusion of motion. There are two background available; an outside setting and a dungeon setting (the dungeon setting is intended for harder levels). The background images are loaded through a repeater within the Game module. The Game module is responsible for manipulating the Background object so most of the code is there.

Game

The second half of the game file contains code written by the author. The difficulty is set which affects the character, background and obstacle speed. The obstacle collision is also included within the game, this is one of the more complex functions as the character can actually traverse the top of obstacles instead of causing them damage. This is one of the main functions within the game play.

Obstacle

This module creates an obstacle canvas. These are images of either wooden or stone boxes. The obstacles move across the screen at the same rate as the background. The obstacles are randomly placed within a preset area. This is quite complex code and reduces the chance of an overlap.

EndScreen

This is the screen shown when the game is finished. It allows users to either play again or enter a name and submit a score.

HelpScreen

The help screen details how to play the game, the objectives and how score is calculated.

TitleScreen

This is the title screen for the game, it allows users to:

- Choose difficulty
- Choose character
- Begin playing the game
- View the help screen

ScoreBoard

This is the scoreboard component for the game. It displays the scores currently recorded within the MySQL database.

setScores.php

This PHP script is used to update the scores held within the database. The output is an XML file which contains the new Score ID within the database, this is returned to allow for the score to be highlighted in the score board. The setScores file returns numerous error messages depending on the case – this allows the user to have constructive feedback and if a log file was introduced it would be easy to see what has happened.

getScores.php

This PHP script is used to return the scores held within the database. The output is an XML file containing names, scores and dates. The getScores file returns numerous error messages depending on the case – this allows the user to have constructive feedback and if a log file was introduced it would be easy to see what has happened.

Database

The database contains data for the scores; a username, score and date for each record. The database is relatively simple but created without reference to the course material.

What I have gained from this project

This project has enabled me to utilise new environments. Both environments are cutting edge and used in industry so it was nice to get a taste of what we could enter into when we graduate. It would be good if more modules used newer technologies as it creates new challenges and allows the output to look a lot better. I am genuinely proud of what I accomplished in both modules. The 3D modelling was something completely new to me and I have not used a tool as powerful as 3ds max before. I also found the Flex project's requirements expandable enough so that I could challenge myself and produce a really nice looking game.

Were I to complete the project again, I would definitely want to expand the Flex/PHP/MySQL connectivity as it interests me. I also enjoyed creating moving objects and writing functions to collect collisions so would perhaps include more of those.

Physics has always interested me so the (albeit simplified) forces within the module interested me and I enjoyed implementing them within the project.

One aspect I was disappointed with was the confusion between when to use MXML and when to use ActionScript. I feel as coders we should always be using ActionScript but that may detract from the ability to create applications rapidly.