|  |  |
| --- | --- |
| Christopher Dye | DYE11210783 |
| Nick Elson | ELS09170089 |
| James Brown | BRO09194788 |
| Jake Boud | BOU11209172 |
| Tom Pendle | PEN11204467 |
|  |  |
| CMP2068M | |
| Group Project | |
| Amr Ahmed | |

**16/01/2013**

Y

Game Design Document

# Progenitor

# 

Chris Dye

Jake Boud

James Brown

Nick Elson

Tom Pendle

Identification of Requirements

Our group project follows the theme of procedurally generated creatures; the player must be able to control a species, nurture and raise them. This could include things like changing their skills and abilities depending on how the player looks after the creature. This would mean that the project we create should be built on code and software that could morph appropriately, as well as give suitable interaction with the user. Most programming languages at the moment support procedurally written code, so being able to create randomly generated creatures is achievable, and would only be as complex as we code it.

The creatures in the game, human or AI controlled must be procedurally generated using a genetic code. From this we know that our player must be able to change what their creature can do, or how it acts, depending on what they do with it. Similarly the AI controlled creatures should also be able to change their own behaviour, and appearance depending on their environment, and how they react within it. To do this will require each AI creature to have the smarts to get around the area it is in, and how it changes depending on that, this could be a tricky one to code, due to the player having no real input on how they live. The AI would also have to provide some challenge to the player otherwise the actual game play would provide no challenge to the user.

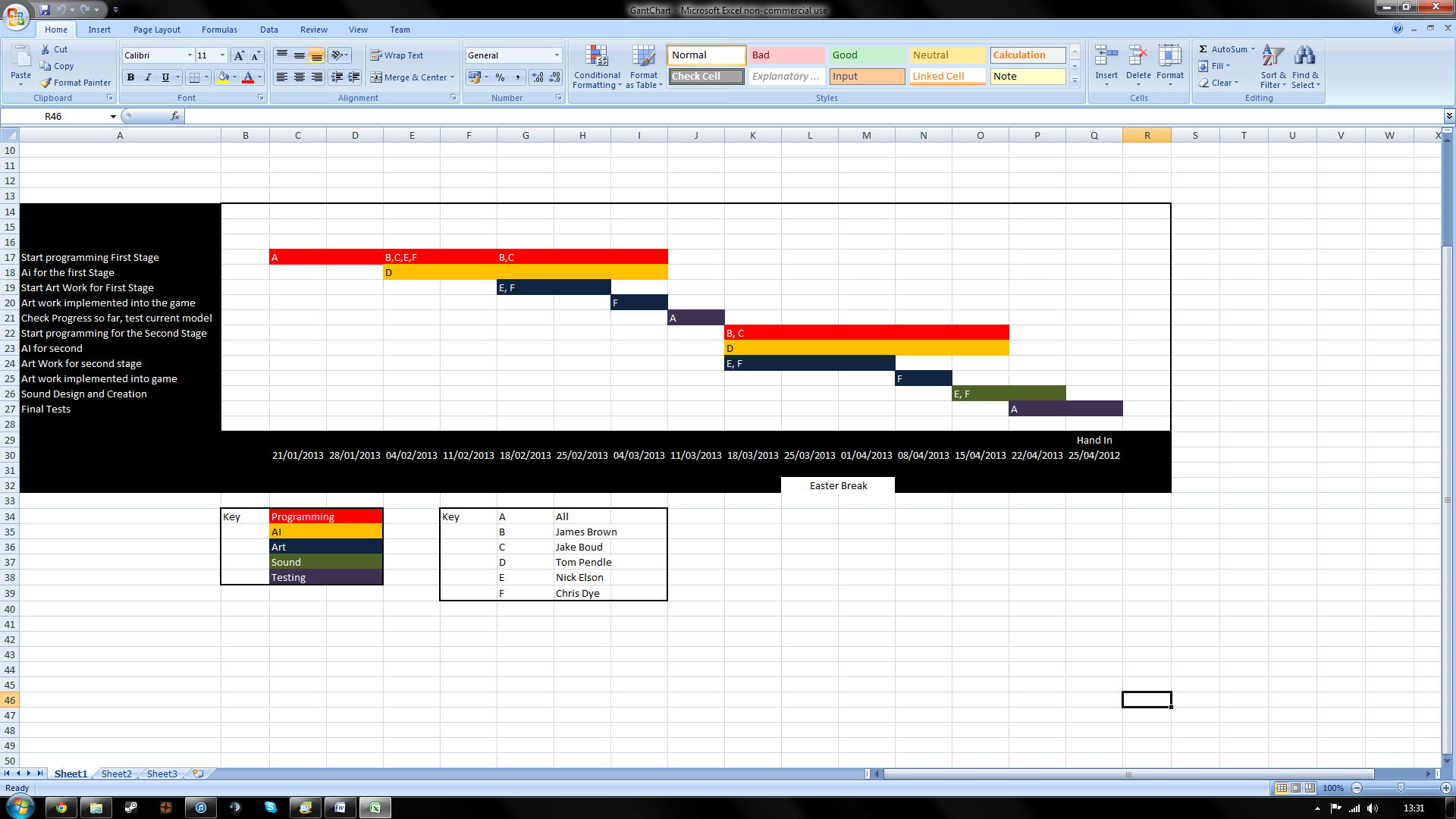
In the game there should be a mechanism that genetically generates unique and new species that can inherit features of others. This will eventually create different creatures through game play, and/or AI interaction. This also means that whenever the game is picked up, or replayed the outcome will almost always be different. The genetic representation should be sophisticated enough to support a large variation of creatures, this could be done by assigning attributes such as strength and defence or health and attack, etc. The random outcome of the game could be coded in a variety of ways, for example random number generators, could power the AI and their interaction with other AI.

The brief also requires a natural selection mechanism so that there is some sort of competition between creatures, this could be for food, or just a fight for survival/dominance. The stronger creatures should be able to pro create and become a more dominant species, whereas inferior creatures or creatures that are to slow to develop will eventually be picked off by superior, eventually becoming extinct. In code this could be done by using collision of the creatures, once two collide the code should be able to distinguish what will happen depending on the situation, the AI will more than likely be the most complex part of the project all together, but will also be the defining feature of the game which fits most of the requirements.

The game should also have a suitable UI, which will relay information to the user so that they can play the game effectively. Most game engines feature a HUD which is easy to implement, and if the software doesn't they are usually easy to find or implement. The main issues we could face with a HUD is laying out information in a way that easy to understand and provides the information necessary to progress.  
 The final artefact must feature all of the features mentioned previously to suit the clients requirements. The main issues the requirements creates is the way we implement AI, and how the players own creature will grow and develop, but procedurally generated code is easily achievable now a days with modern programming languages. AI can be coded as complex as we make it so this should also be possible for us to overcome any problems we face when creating it.

## Milestones

The Gant Chart (Also given in a separate file) explains our timeframe for completion of the project. We have given ourselves a week mid way through the project so we can look back on our project to see if we are making good progress so far, and see if there are any changes we need to make. The programming and AI will almost always be done simultaneously, because generally they will work of each other. Some tasks, like the implementation of the art, cannot be done until certain things are finished in the main program, like the characters, however as we will use sprite sheets implementing them shouldn't take too long.



Overview

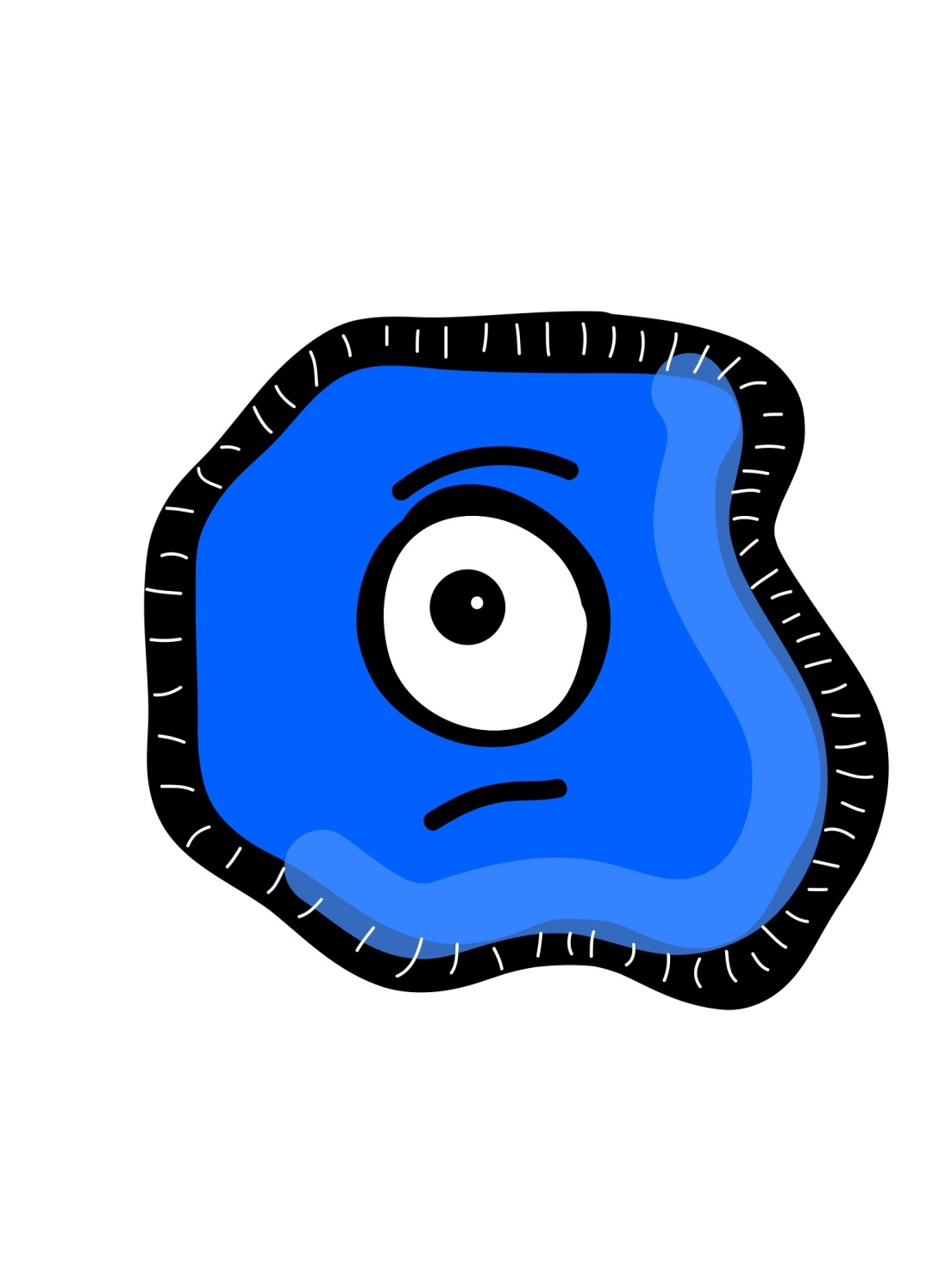
Introduction

This third person interactive role-playing game allows the player to experience the evolutionary cycle of a new species, from a single cell organism all the way to a fully evolved creature. Throughout this journey the player will make decisions about how the creature interacts with other creatures and mould its evolutionary path, changing its appearance and genetics.

## Detailed Game Description

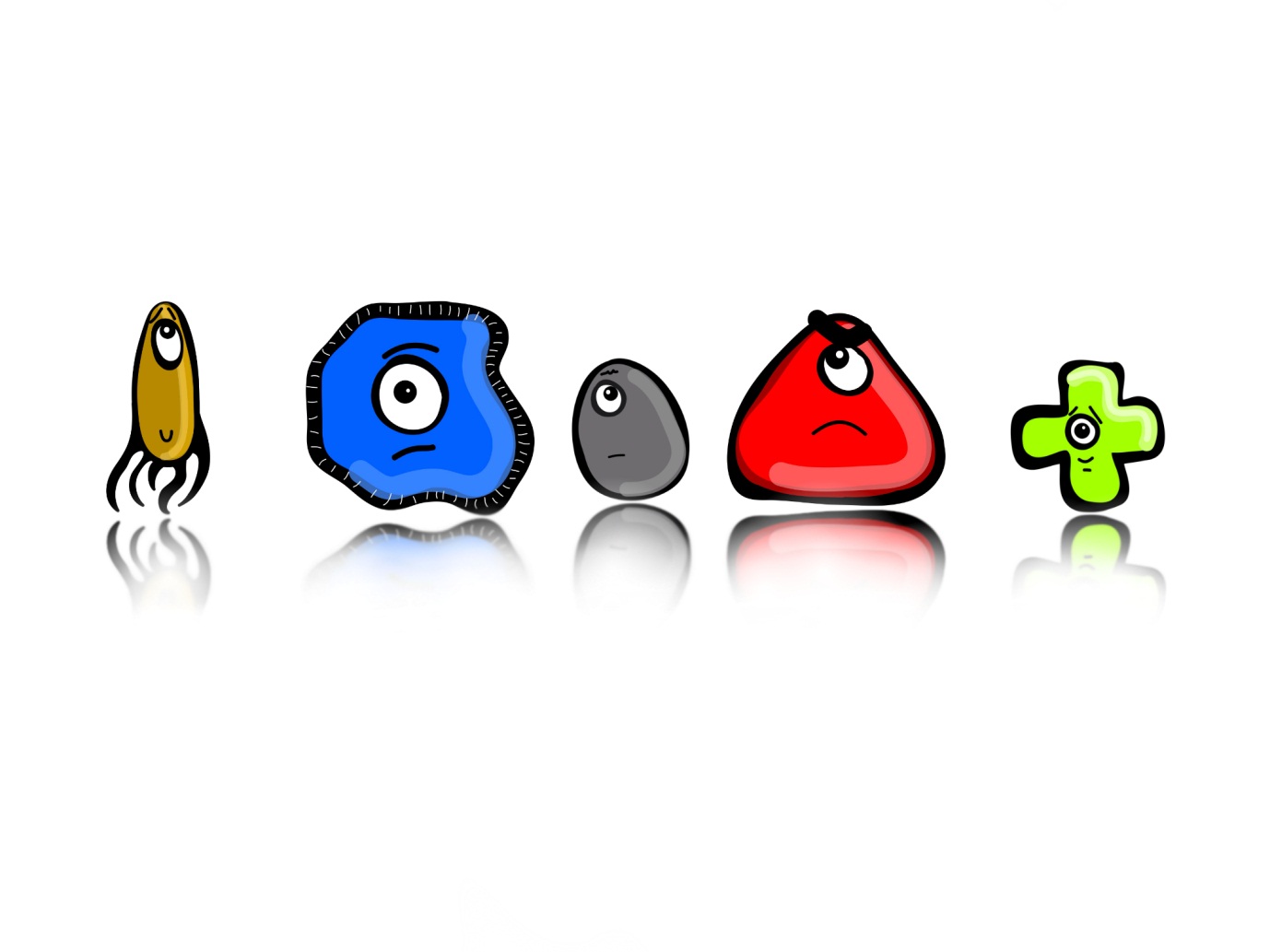
Concept

The high level concept for Progenitor is a fast paced simulation game where the player moulds their creature to become the strongest by battling other creatures and breeding to create the ultimate evolutionary creature.



Characters

The main character in the game is the player’s own creature and the player will see the game world from their perspective. This creature will be fully controlled by interaction from the player, and will become stronger and more powerful during the game.

All of the NPCs are other creatures in the game world. These creatures are on an evolutionary journey as well, and they will interact with the player’s creature as well as each other. There will be battles between creatures to decide who is the stronger, and creatures will also be able to breed. The player can choose to fight with the other creatures in the world as well.

Story

Starting off as a single-cell organism, our creature must evolve in order to gain dominance over rival creatures. Consuming other, weaker organisms will allow our creature to become stronger and bigger, until it becomes powerful enough to grow to the next evolutionary stage.

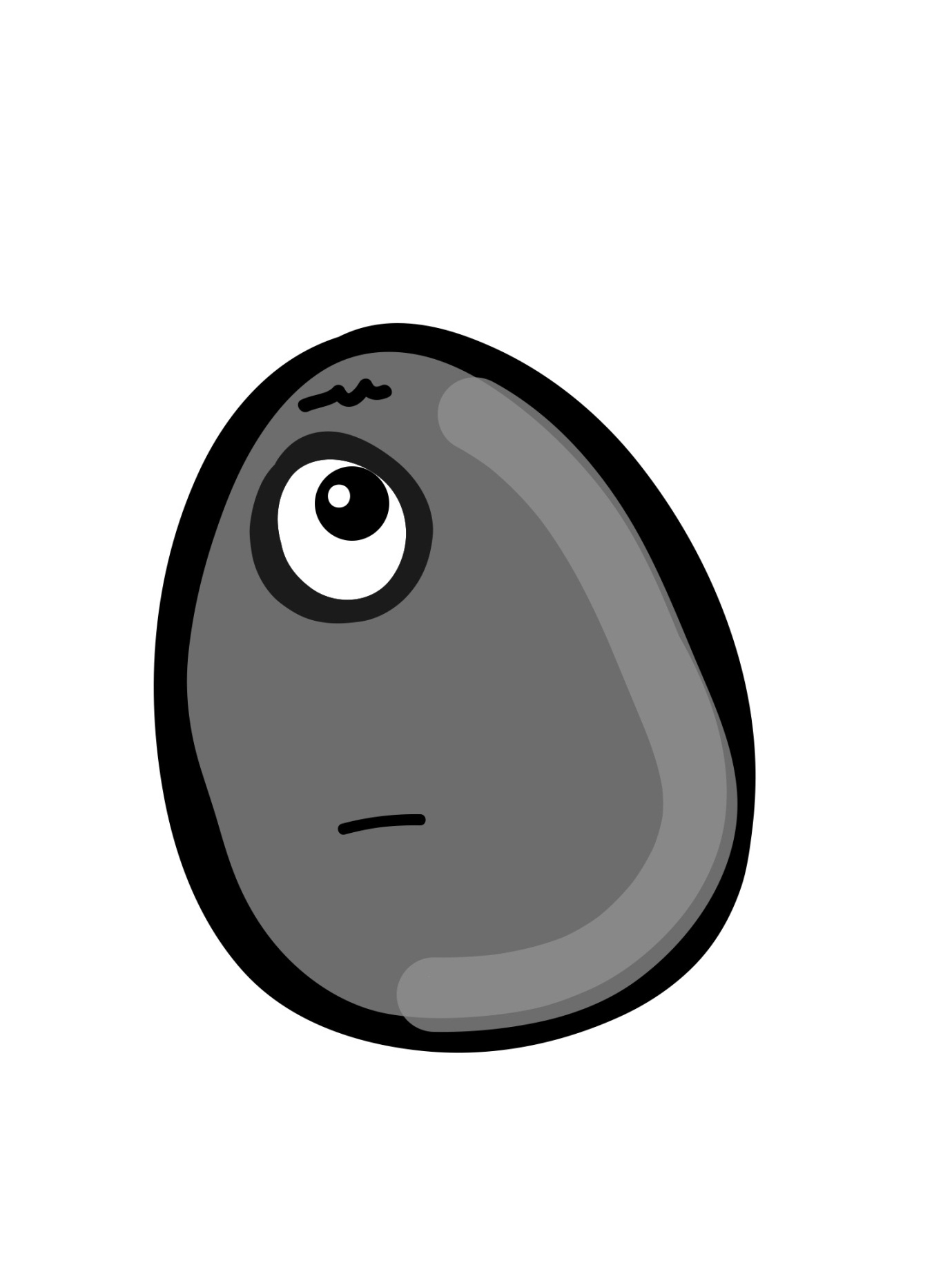
Once evolved, our creature will be able to observe the actions of its various rivals and decide how to interact with them. By battling with others, our creature will mutate and grow its own claws, legs, wings and more. Will our creature overpower all others and become the most feared? Will it survive long enough to reproduce and create an even stronger heir?

Objectives

The main objective of the game is therefore to make the strongest possible creature, by defeating other players and gaining core attributes. Whilst the general focus of our game is allowing the player creativity to mould their own creature, it must be a strong creature too, so the player must compromise style with practicality.

Alongside this is our secondary objective; for the player to survive long enough to reproduce. During the ‘end game’, the player will be given the chance to breed and produce a child, which will inherit some of its attributes. This player can then play the game through again with this child as their creature.

Due to the fact that our game is designed to be played again and again with subsequent generations of creatures, there is an over-arching tertiary objective, which is to complete subsequent generations better, or faster than you have before. The difficulty of the game will however increase each play through, so even though your creature should be stronger, its rivals will be harder to defeat too.



Gameplay

Our game is broken into two stages, the cell stage and the creature stage. At the start of the game, the player begins as a single cell organism. This cell has four key attributes:

* **Health (Green)**
* **Attack (Red)**
* **Defence (Blue)**
* **Speed (Yellow)**

These attributes will form the backbone of the whole game. Cells which are of different sizes to the player’s will appear from the edges of the screen sporadically, and these will be colour coded to match the aforementioned attributes. Our cell is controlled by the player and can move around freely, until it comes into contact with another cell. If the other cell is smaller than ours, we consume it and 1 point will be assigned to the attribute it represented. However, if the other cell is bigger, we will be eaten and have to restart this stage.

The purpose of this stage is to allow the player to assign attribute points to their creature, which will carry through to the start of the next stage. The player has 100 points to assign any way they choose (by consuming other cells), and once all 100 have been assigned, the cell will evolve into our creature, and the second stage will begin. It is important to note that during this stage, the colour and appearance of our cell will change dramatically as a visual indicator of its attributes (a cell with 100 speed points assigned only would be yellow, more streamlined in shape and would move more quickly around the screen). Our cell does start with a speed advantage over the others to allow us to catch and consume them at the beginning of the stage. The other cells will also chase and consume each other.

Once we enter the second stage, we will be in full control of our creature, and this is where the bulk of the game will be played out. Our assigned attributes from the first stage are carried over, and will change the physical appearance of our creature. We can direct our creature around the game world and interact with other (randomly generated) creatures. These creatures are all hostile, so it is the player’s decision if they decide to approach them or not, based on their physical appearance which again will be representative of their attribute values.

If the player decides to interact with another creature, a turn based battle scenario will commence, where the attributes of both our creature and the opponent creature will be taken into account. The player will choose from the following options:

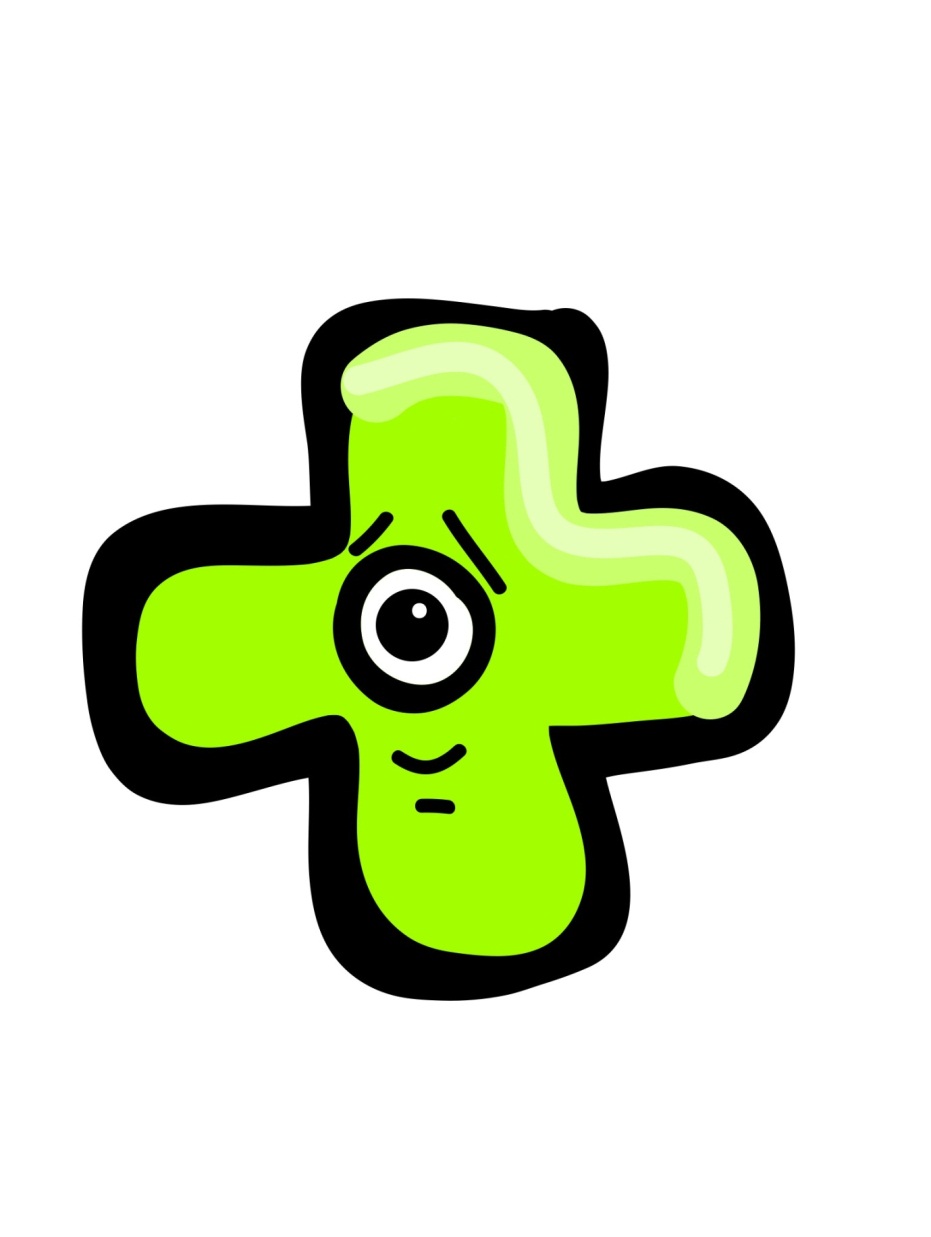
* Attack
* Defend
* Escape

Damage dealt each turn will result from the attacker’s ‘Attack’ attribute value minus the defender’s ‘Defence’ attribute value and will be taken off the receiver’s ‘Health’ value. Once one creatures health is reduced to zero, they are defeated and the other creature victorious. If the player wins, they will gain evolution points, and have a chance to earn a particular physical feature of their opponent. ‘Defending’ will raise the player’s defence attribute for the rest of the battle, menacing our creature will take less damage for the rest of the turns. Choosing the ‘Escape’ option during the battle will result in the battle finishing only if the player has a higher ‘Speed’ attribute then their opponent, otherwise they will be unable to flee; putting a great emphasis on the player making an educated decision before approaching another creature.

The main objective in this stage is simply to survive until the end of the game, whilst developing your creature and gaining levels through evolution points. Every level that your creature progresses will grant them extra attribute points based on the creatures they have defeated. For example, if the player has defeated a lot of fast creatures, the majority of these points will be assigned to speed. For every five levels that your creature grows; there will be a special battle against a strong opponent, defeating this creature will award the player with a checkpoint and a special, game-changing physical feature.

Once a player reaches level 30, they will fight the strongest opponent in the game. Winning this final battle allows the player to become the strongest creature in the game. At this point, our creature (being the alpha male) breeds and produces a child that inherits some of its attributes. The player will now play as the child creature from the beginning of the creature stage, with a slightly higher difficulty level, but the opportunity to become an even stronger creature than before.

Once the game has been played through more than twice (using ‘new game’ option), and the player completes the game they will have the opportunity to breed with one of their previous creatures. This allows the player to more accurately choose which mix of attributes will be passed on to the next generation of creatures, emulating the real world of natural selection, where two strong breeding partners should produce a strong child.



## Game Structure

Interface

The game will load up to a title screen with the options:

* New Game
* Load Game
* Family Tree
* High Scores
* Options
* Exit

These are to start a new game, continue from an saved game, and view a ‘Family Tree’ of all the creatures you have completed the game with. The high scores page will show scoring statistics for the player.

Whilst playing the game, the player will be able to pause the game and access a screen where they can see their creature’s attributes and various statistics regarding battles won/ levels gained, etc. and a colour wheel showing where our creature lies between the four attributes.

Graphics will be one of the most important features of our game. We are implementing a unique art style that will allow the player to instantly tell the attributes of a creature/ cell upon first glance. This will use both the colour of that creatures colour wheel as well as specific physical features combining to give an instant indication of how strong an opposition it will be, and more importantly whether or not our players creature is wise to attack it or not.

Controls

The controls will be an implementation of the keyboard buttons. ‘W’, ’A’, ’S’ and ‘D’ will control the movement of the cell/ creature. ‘P’ will pause the game, and the arrow keys will navigate the menus and title screen, with enter being used to select the currently highlighted option.

Scoring

Each completed game will be scored on three categories:

* Time taken to complete the game
* Number of enemies slain
* Number of generations completed

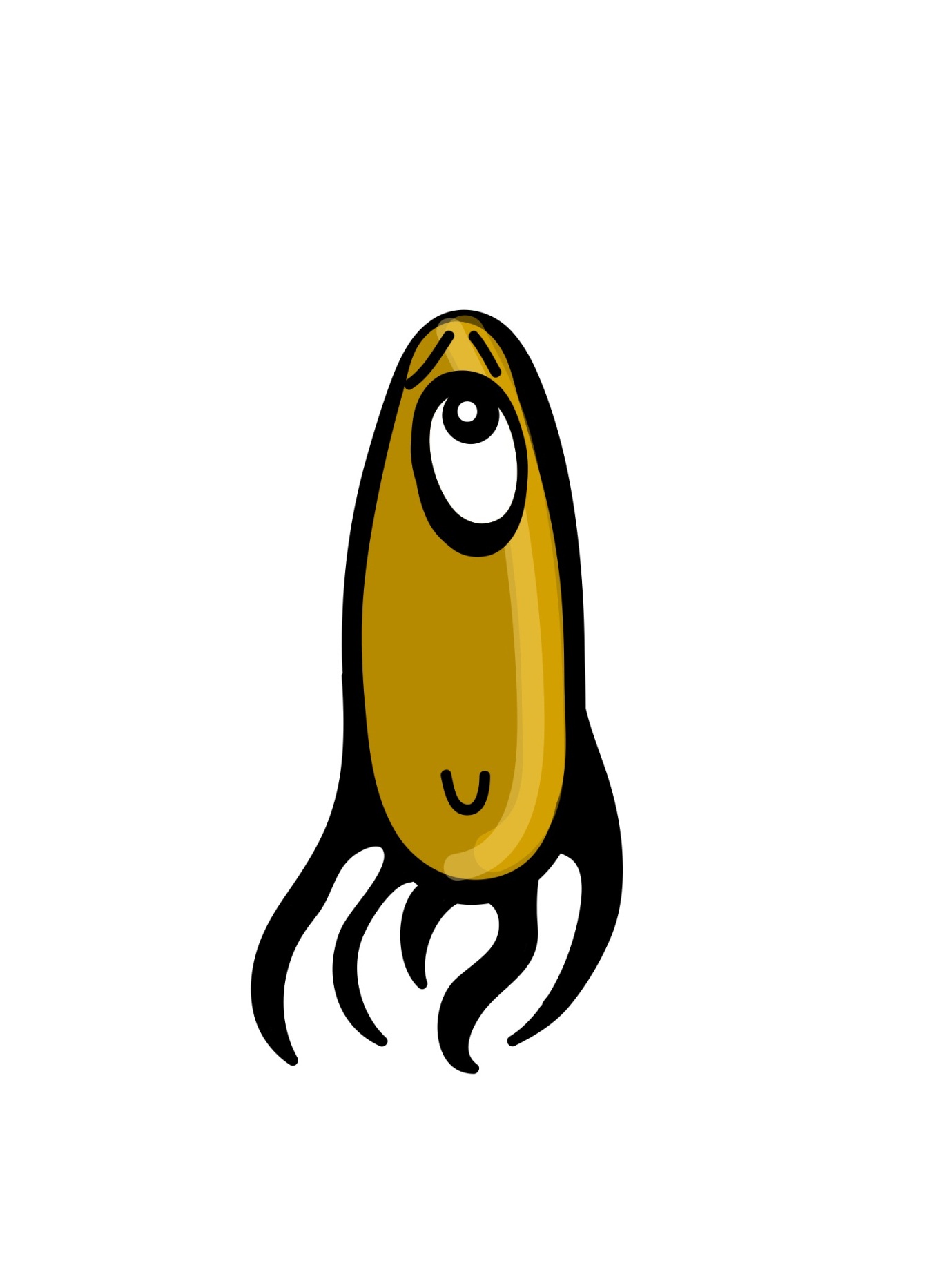
These scores will be visible on the high scores screen accessed from the main menu.

Levels

There will be two main stages, the cell stage and the creature stage, so these are the two levels in the game. In the creature stage however we will implement slightly varying level design in terms of scenery that will change throughout the game, almost like the different seasons to show a passage of time.

Audio

The audio in the game will consist of several pieces of background music that will be constantly played throughout the game, and then will be overlaid with contextual sound effects to add to the player’s immersion in what is happening.

****

Identification of risks and resources inherent within a group project

This section will include all risks associated with working in a group on a project, including those which affect requirement gathering, project planning, implementation and delivery phases of the project. Moreover it will identify the resources that are available and programmed into the project plan, including human and technical resources.

Risks

Human Risks:

* Illness:

This risk will be a potential problem throughout the whole project and will be difficult to avoid as it is hard to predict if someone will become ill. It could be a potential problem if they are unable to work on their stage of the project which is necessary to be finished for other stages to begin, it could be rectified by requesting for an extension.

* Inability to understand how to use the toolset:

For example if a specific group member couldn’t use Unity effectively, this could also be seen as a technical risk and will be mainly an issue in the implementation and delivery phases of the project as it involves the actual creation of the artefact, this problem could be rectified by changing which stage of the project they work on.

* Non-engagement in the project:

This could cause stages of the project not being completed on time and as a result stall the starting of dependent stages that follow them, this issue could be rectified by motivating the group member isn’t engaged in the project.

Technical risks:

* Hardware failure:

This could lead to a group member being unable to work on the project, as a result it could delay the project as the stage they are working on could be necessary for another stage to begin, to rectify this issue they will have to find an alternative way in which they can work until the problem is solved.

* Software problems:

This could make it so the group members are unable to work on the project, such as an update which makes it so the program does not work anymore, this will mean that the group members will either have to go back to a previous version of the software or wait for it to be fixed, this means that they could be delayed in finishing that stage of the project, which could delay other stages of the project which require that stage to be completed.

Human Resources

The human resources that will be available during the project will include each group member’s specific skills and experiences that will enable them to perform certain roles within the project. Below is listed each member of the group with a description of their skills and experiences that will help them in the project.

Christopher Dye:

Strengths:

* Willingness to work hard on tasks that he is given and skill in artistic applications, this means that he will be able to support others in doing tasks which they are more specialised in and help complete aspects of the project which do not require a specific specialisation.

Weaknesses:

* Not particularly strong coding skills and a lack of recent experience using Unity.

Jake Boud:

Strengths:

* Very strong mathematical, logical and problem solving abilities
* Good programming skills and experience, this means that he will be able work on the core programming tasks as well as helping to come up with logical solutions to problems that affect the project as a whole.

Weaknesses:

* A lack of artistic skills and a lack of recent experience using Unity.

James Brown:

Strengths:

* General programming abilities and organisational skills
* Recent experience of using Unity, this means that he will be able to help the other programmers get to grips with using Unity again so that they can all get up to speed with creating the artefact.

Weaknesses:

* A lack of experience in AI programming as well as not particularly good artistic skills.

Nick Elson:

Strengths:

* Artistic and creative abilities, this means that he will be able to create the graphics and some sounds for the artefact and focus on the overall visuals of it.

Weaknesses:

* A lack of understanding of mathematics.

Thomas Pendle:

Strengths:

* Strong mathematical, logical and problem solving abilities
* Experience of AI programming, this means that he will be able to concentrate on the AI and core programming of the project.

Weaknesses:

* A lack of artistic skills.

Technical Resources

The technical resources that will be available during the project include both hardware and software resources.

Hardware:

* Our own computers/laptops for development
* Mobile phones, for communication
* The university computers in the computing labs for development

Software: that we have decided to use includes

* Unity as the game engine
* The programming language we have decided to use is C#
* The group tools we have decided to use is GitHub

Contingency Plan

Loss of human resources:

* If one of the group members is unable to continue with the project, we will then divide their workload amongst the other group members so that the project can continue

Failure to complete a stage of the project:

* If a vital stage of the project is not completed we will request the assistance of the project supervisor

Toolsets

The toolsets that our group shall be using will be focused on the best tools to create a game and ways to facilitate easy sharing and collaboration. We considered many game engines, programming languages and sharing applications. After looking at the available options the choices we made were to use Unity as the game engine and maker, C# as the programming language and Git/GitHub to share and collaborate.

# Engines

Gamemaker

Gamemaker was an obvious consideration because all members of the group have used the software last year for assignments which means it would be easy to start with. Together with our own familiarity Gamemaker is very easy to use so we would be able to create a prototype very quickly, however the features that make Gamemaker easy to use also hinder its ability when making more complex games. It lacks the ability to use a complex/widely used programming language which means it has a limited amount of uses, the main example being that 3D is very hard to implement. And though it has an easy to use sprite based graphic system other types of graphics would more difficult, such as our planned vector based graphics. The very restricted free version also means any game we make using it would have a permanent watermark which is not very good when showing the game to an audience, the free versions of the other engines have no watermark.

#### For

* Easy to use
* Prior knowledge of use by all
* Simple sprite based graphics

Against

* 3D difficult to implement
* Limited programming language
* Restricted free version

XNA Game Studio

Another game engine that we considered was XNA Game Studio as it supports a popular language (C#) which is also the language that the group is best suited to. Together with its use of C# XNA Game Studio uses Visual Studio for programming which the whole group has much experience with, but none of us have used XNA Game Studio itself so learning it would take longer than other alternatives. It also has a heavily code based environment that does not help when learning the software itself or making quick prototypes, issues that are vital to this project. Finally it has very limited platform support, only supporting Windows and Xbox 360.

#### For

* C# is supported
* Free commercial license
* Integrated with Visual Studio

Against

* Unfamiliar
* Heavily code based environment
* Only runs on PC and Xbox.

UDK

The Unreal Development Kit (UDK) is a popular tool for developing games and levels, its open license means it is easy to sell a game once it is made. It also has a lot of documentation and 3rd party guides and tutorials to help people use the engine easily, though this is somewhat offset by the rather complex and confusing interface. This could affect the workflow of the group as some tasks could take a while to complete if they need to be heavily researched beforehand. Also UDK uses a custom scripting language for in game coding called UnrealScript, as it is different to the standard C++/C#/Java it will also require extra time to learn and use initially. Though the engine has problems in the using and familiarisation areas it can create very high quality graphics quite easily which is good for many games, however our game is a 2D/isometric one that would not benefit hugely from such visuals.

#### For

* Popular, well documented
* High graphic quality
* Good free license

Against

* Uses custom programming language
* Confusing interface/slow workflow

**Cryengine 3**

Like UDK Cryengine 3 also has a very high quality graphics engine which again is not 100% useful for our project, it is also presented in a development kit of sorts, however the user interface is easier to understand than UDK’s. Though as it is a newer and lesser used engine it lacks the amount of documentation and guidance that UDK has which makes it troublesome to use as no one in the group has ever used it. Like XNA Game Studio and UDK it has a generous free license but as it is also very weighted to 3D games it is not useful for us to use.

#### For

* Very good graphical quality
* Good interface
* Good free license

Against

* Not hugely popular
* Suited mainly for 3D games

**Unity**

Unity is the game engine/development kit that our group has chosen to use for the project. There were many reasons for choosing this over the others mentioned here, one of the main reasons was that most of the group have had experience with Unity as it was used at the end of the 1st year for workshops. This means that we can instantly start using the software instead of having to learn its basic functions from the ground up. Also Unity is itself easy to use to build both a full game and prototypes, which is perfect for our purpose. The license is as good as most of the options here allowing full use and commercialisation, though the free license does not have as many features as the paid for version the missing areas are mostly graphical which is not of much concern for us. One concern is that the project files and folders are reasonably difficult to sync and share via a CVS such as Git as they are large and split, there is a paid for program that helps with this but it is too expensive to consider.

For

* **Easy to use**
* **Free commercial license**
* **Graphic based workflow**

Against

* **Large projects to sync of CVS**

When choosing our game engine we considered carefully the pros and cons of the five choices that we narrowed down to. Gamemaker was quickly discounted as it was far too restrictive in terms of license and coding ability. XNA Game Studio was too unfamiliar to be useful, UDK and Cryengine 3 had similar issues in terms of being too suited to 3D games and the high graphical quality was by no means needed. Unity fitted what we needed very well; it is easy to use, familiar and lets us have flexibility in terms of output platforms.

# Programming Language

As we had chosen Unity as the game engine there was only a choice of two programming languages that are available to use, a version of JavaScript called UnityScript and C#. They both have their own pros and cons in the context of developing a game in Unity.

JavaScript/UnityScript

#### For

* Some experience
* Simple to learn
* Used in many other applications

#### Against

* Not much experience
* Type used in Unity is slightly different
* Must use OOP

A version of JavaScript, UnityScript, is one of the two programming languages in the Unity engine so it and C# were at the top of the list of languages to use. UnityScript can be useful as it has a simple syntax that isn’t too alien compared to other languages, also many Unity tutorials use JavaScript as the main example with C# being less used. Though it is simple to learn the type of language used in Unity is not the same as the language used in JavaScript in webpages and other applications. Also our group has only had a small amount of experience in JavaScript which was in the use of databases compared to a lot of use of C#. Also UnityScript can be slower in execution compared to C# as it does extra work in areas where C# would not, like when using the dynamic typing feature in UnityScript.

**C#**

For

* **Many other uses**
* **Known to all group members**
* **More documented than JS, outside of Unity.**

Against

* **Not used in majority of games**
* **Less efficient than C++**
* **Not compatible on non-windows platforms.**

C# is our chosen language for the project, the main reason being that every member of the group has had a lot of experience with C# and its family members such as C++ and C, this means we can start work on the project as soon as possible without having to wait while we learn the ins and outs of a new or unfamiliar language. C# also has many uses outside Unity that we will know more of by the end of the project just because we are using the language regularly, and although UnityScript is included in more Unity tutorials C# is covered far more extensively outside it than Java/UnityScript is. C# does however have some disadvantages, one is that it is not used as much as languages like C++ in game programming, however C++ is not supported in Unity and there are many similarities between the two that offset this disadvantage. Another is that you must use OOP to make you code work properly so if some group members had not known this it would have to be learnt.

From these two languages we chose C# as it offered a near non-existent learning curve for all members compared to a pretty steep one for UnityScript, and the advantages of more tutorials for UnityScript did not outweigh this and the wider application of C# when programming games.

# Group Tools

The group tools we will use while making the game are a very important part of the whole project as without proper ones we would not be able to work together efficiently as a team, which would lead to mistakes and errors that could slow us down or worse.

**Github**

For

* **Has very good versioning**
* **And branching**
* **Works on non-mobile platforms**

Against

* **Complicated**
* **Not on mobile platforms**

One of the choices we had was the distributed revision control and source code management (SCM) system Git, or in our case the hosting service GitHub. GitHub allows for quick and easy uploading, merging and forking with a cloud based server, it is also free for non-commercial use. This means all means all members of the group can have their own fork of the master copy that can be combined when meetings occur, the forks mean that any changes members make won’t accidently overwrite other peoples work and that there is always a backup unchanged copy that works in the master. It is also well used in the workplace according to Ohloh.net 29% of professional software developers use git, up from 13% in 2011[[1]](#footnote-1). For people who have never used a SCM system it can be quite complicated in terminology and interface, however GitHub is rather user friendly and has a non-command prompt interface. Though it has no mobile app that allows downloading and uploading of repositories.

**Dropbox**

#### For

* Free for up to 10gb per person
* Available on all platforms
* Easy to use

Against

* Has no versioning
* Or branching
* Limit on storage

Dropbox was another choice for a group tool, it is a very popular cloud storage solution that is available on nearly every platform, and it is easy to use and free for up to 10 GB per person. It allow for folders of files to be shared between users to allow group editing and downloading of files. However it is not a revision control system and does not easily allow backups and versioning of files, or the forking of files from a master copy. These features are vital in a complex group project like this one, and the limit of 10 GB means many copies of the source code and data of a game would not be able to fit. So although it is very easy to use the lack of complex features means it was not suited to our needs.

There are also SCM systems other than Git however but these are either client-server models which would not work for our group as getting a server would be far too time consuming and expensive. So we are restricted to other distributed revision control systems but Git is by far the most popular and well supported so this was the logical choice for our group.

1. <http://www.ohloh.net/repositories/compare> [↑](#footnote-ref-1)