Game Development Assignment 1

# Task 1

* *Describe the development process of small sized games/applications*

Steps for game development process:

* Research: Once an idea about a clicker game rose, I did some research into clicker games and played some of the cookie clicker game. This helped to form the idea and concept around what dynamics and mechanics clicker games usually employ.
* Concept: Following the research, I formulated a concept for my own clicker game. The general concept was a clicker game in which the player would be attempting to keep citizens of a population from dying, as they would have a short lifespan.
* Design: The design that stemmed from this concept was that clicking brought to life a certain number of populace members. The game reduces an amount of populace members every specified amount of time. The upgrades available would increase the members’ inherent life span or would give the player additional populace members per click.
* Implementation: I will be using Unity Engine to implement this design using C# object oriented programming language.
* Testing: Finally I will be testing the game before creating a build to be released and submitted for grading.
* *Identify common game engines and programming languages in the industry*

1. Game Engines:
   1. Unity – C#
   2. Unreal – C++
   3. Godot – GDScript, Python, C#, C++, VisualBasic Script
   4. CryEngine 3 – LUA, C++
2. RPG Maker MV – Java

In conclusion, many common programming languages for game development are OOP languages, as the concept of objects in this paradigm are synonymous with game objects and how they relate to each other.

* + *Compare the languages*

|  |  |  |  |
| --- | --- | --- | --- |
| **Language:** | **C#** | **Python** | **C++** |
| **Typecasting** | Static | Dynamic | Static |
| **Programming paradigm** | Object Oriented | Multi-paradigm (OOP, Procedural) | Multi-paradigm (OOP, Procedural) |
| **License** | Closed, Microsoft licensed | Open-source | Open-source (depending on implementation) |
| **Comments** | // or /\*\*/ | # | // or /\*\*/ |
| **Line endings** | ; | No line endings | ; |
| **Method bodies** | Encased in { } | Methods are not encased in any characters and are instead read in interpreted flow, signified by a colon after the function name | Encased in { } |
| **Namespaces** | Explicitly typed | Not required | Explicitly typed |
| **Variable naming** | Cannot use keywords or system definitions | Cannot use reserved words | Cannot use keyword or system definitions |

* + *What is different in how their syntax is written*

(See above table for more info.) Essentially, due to the nature of typecasting within different programming languages, Python can be written in much less code compared to C# or C++ as a lot of the code is interpreted by the compiler at runtime, whereas the other two are interpreted prior to runtime. C# and C++ are also strongly typed, meaning that you cannot have variables without specific datatype definitions and those variables can never be any other data type.

Another difference in syntax is both C# and C++ declare their method bodies within curly parenthesis { }, whereas with Python the method body is simply any following line after a function name declaration and colon. Example:

My\_Function():

#my

#method

#body

Namespaces are also not usually required syntactically with Python as values and variables are interpreted at runtime, whereas C# and C++ will require you to be specific and diligent with including namespace declarations when using code from said namespace.

* + *How are scripts structured*

In Python and in C# both programs start with a method or function called “Main()”, and similarly in both languages they usually communicate between ‘modules’ of code. In Python those look like different functions and vague structures of functions, and in C# those modules are much more strictly defined in containers such as namespaces, classes, and structs. That’s about where the script structure similarities end.

C# structures its scripts with its syntactically required namespace declarations, then what namespace is within the script file (or multiple namespaces after each other, though 1 per file is industry standard) and classes within that. The first of this code to be run is the class called ‘Program’ if the scripts are not using a game engine, in which case the flow of scripts are dependant on how the objects interact. Otherwise, the script will start with a function/method called Start(). A method called Update() will run every frame with most game engines.

Python however, controls its flow mostly through the order in which the functions and script calls are written. There is the Main() function where a script will start if outside the realm of a game engine, but otherwise a module would have to be written to loop through other functions or modules.

* *Explain the data types you will be using and why*
* **Long**: for storing population counter.
* **Int**: for storing indices of arrays
* **Float**: for determining the life (in seconds) of a population member
* **String**: for storing upgrade names and population count display
* **Image/Canvas/Button/GameObject**: complex data type within the UnityEngine. Used to display clickable buttons, render objects on screen, and display or manipulate any graphical items.
* **Array**: an array is a similar collection of elements of the same data type. It will be used for keeping track of the upgrades stored.
* *Explain what mathematical operations you can use to calculate your equations.*
* **Plus** (+, addition): for use with increasing the number of the populous per click, adding to the lifetime of the population, tracking the indices of arrays, for loops, and string concatenation and conjunction.
* **Minus** (-, subtraction): for use with decreasing the population over time.
* **Times** (\*, x, multiplication): for use with exponentially increasing populous added per click, for use with increasing population in houses.
* *Explain any mathematical terms you may use in the document.*
* **Algorithm**: a procedure or set of steps used to solve mathematical calculations or computations.
* **Decimal**: a fractional real number.
* **Integers**: a whole number that can include zero and be even or odd.