CONSTRAINED DEVELOPMENT TASK

Version 1.0
BSc Computing for Games
COMP310

Dr Ed Powley

Introduction

In this assignment, you are required to **design** and **implement** a "de-make" of a well-known game. Your de-make will be implemented in assembly language and will run on the Nintendo Entertainment System (NES).

In the days of the NES, game programmers used assembly language to give them fine low-level control over extremely limited hardware. The experience of developing a game in this way will give you an intimate understanding of how a typical CPU and machine architecture function at a low level, and experience of using this architecture to your advantage, which are useful skills even when programming in high-level languages for more powerful hardware. In addition, a de-make is a focussed exercise in game design, which requires you to focus on core mechanics rather than content and non-essential features.

"Because of the nature of Moore's law, anything that an extremely clever graphics programmer can do at one point can be replicated by a merely competent programmer some number of years later."

— John Carmack

"Programming is like this amazing puzzle game where the puzzles are created by your own stupidity."

— Brad O'Farrell

This assignment is formed of several parts:

(A) **Prepare** a 10-minute presentation that will:
  (i) **outline** the concept of your de-make, including the game it is based upon;
  (ii) **explain** the key mechanic around which your de-make will be based;
  (iii) **justify** the technical feasibility of your concept.
(B) **Populate** a Trello board with the user stories for your demo.
(C) **Implement** a draft program in 6502 assembly language that will
  (i) **implement** your de-make concept.
(D) **Implement** a final program in 6502 assembly language that will
  (i) **address** any shortcomings identified by tutors and peers in Part C.
(E) **Present** a practical demo of your game and source code that will:
  (i) **demonstrate** your academic integrity;
  (ii) **demonstrate** your individual programming knowledge and communication skills.

Assignment Setup

This assignment is a **programming** task. Fork the GitHub repository at the following URL:

`https://github.com/Falmouth-Games-Academy/comp310-portfolio`

Use the existing directory structure and, as required, extend this structure with sub-directories. Ensure that you maintain the `readme.md` file. If necessary, please ensure that you add any editor-specific files and folders to `.gitignore`.

Halo 2600, a de-make of the Halo series developed for the Atari 2600 by Ed Fries.

Part A

Part A consists of a **single formative submission**. This work is **individual** and will be assessed on a **threshold** basis. Answer the following questions to pass:

- What is the title of the game?
- On what well-known game is it based?
- What is the core mechanic that your game will implement?
- Is the game technically feasible, given the limitations of the target platform? Make reference to existing NES games and to technical documentation.
- What is the intended aesthetic?
- Is the scope appropriate for the product development time-frame?

To complete Part A, prepare a 10-minute presentation. You may use any presentation software and visual aids that you deem appropriate.

Deliver your presentation in class. You will receive immediate **informal feedback** from tutor and peers.

Part B

Part B consists of a **single formative submission**. This work is **individual** and will be assessed on a **threshold** basis. Answer the following questions to pass:

- What are the user stories for your demo?
- Which user stories are required for a minimum viable product, and which are stretch goals?
- Is the scope appropriate for the product development time-frame?

To complete Part B, populate a Trello board with your user stories.

Show the Trello board to your **tutor** for immediate **informal feedback**.

Part C

Part C is a **single formative submission**. This work is **individual** and will be assessed on a **threshold** basis. The following criteria are used to determine a pass or fail:

(a) Submission is timely;
(b) Enough work is available to conduct a meaningful review;
(c) A broadly appropriate review of a peer's work is submitted.

To complete Part C, prepare a draft implementation of your game. Ensure that the source code and related assets are pushed to GitHub and a pull request is made prior to the scheduled peer review session. Then, attend the scheduled sprint review session. You will receive immediate **informal feedback**.

Part D

Part D is a **single summative submission**. This work is **individual** and will be assessed on a **criterion-referenced** basis. Please refer to the marking rubric at the end of this document for further detail.

To complete Part D, revise your program based on the feedback you have received. Then, upload a `.zip` file to LearningSpace containing the following:

(a) The source code for your game;
(b) Any art or sound assets required to build your game;
(c) Brief instructions describing how to compile your game from source;
(d) A compiled ROM image of your game suitable for loading into an emulator.

Please note, the LearningSpace will only accept a single `.zip` file. You will receive **formal feedback** three weeks after the final submission deadline.

Part E

Part E is a **single summative submission**. This work is **individual** and will be assessed on a **threshold** basis. The following criteria are used to determine a pass or fail:

(a) Enough work is available to hold a meaningful discussion;
(b) Clear evidence of programming knowledge **and** communication skills;
(c) No breaches of academic integrity.

To complete Part E, prepare a practical demonstration of the game. Ensure that the source code and related assets are pushed to GitHub and a pull request is made prior to the scheduled viva session. Then, attend the scheduled viva session.

Additional Guidance

As always, avoid underestimating the effort required to implement even simple software; always consider scope. From the proposal stage, you should consider very carefully what is feasible. This is especially true for this project, where you are working with extremely limited hardware and programming in a low-level programming language. You are strongly advised to focus on a single game mechanic, and to keep your proposed project simple. Focus on producing a basic playable experience, not on adding unnecessary features.

Your code will be assessed on **functional coherence**: how well the finished product corresponds to the user stories, and whether it has any obvious bugs. Correspondence to user stories runs both ways: implementing features that were not present in the design ("feature creep") is just as bad as neglecting to implement features.

Your code will also be assessed on **sophistication**. To succeed on a project of this nature, you will need to make use of appropriate algorithms, data structures, and programming paradigms. Appropriateness to the task at hand is key: you will **not** receive credit for complexity where something simpler would have sufficed. Indeed, in this case, an overly complex solution is likely to run slowly and hence adversely impact functional coherence.

Maintainability is important in all programming projects, but doubly so when working in a team. Use **comments** liberally to improve code comprehension, and carefully choose the **names** for your files, classes, functions and variables. Use a well-established commenting convention for **high-level documentation**. Also ensure that all code corresponds to a sensible and consistent **formatting style**. Hard-coded **literals** (numbers and strings) within the source should be avoided, with values instead defined as constants together in a single place.

As with all assignments on this course, you are expected to display a level of **innovation and creative flair** befitting Falmouth University's reputation as a world-leading arts institution. One approach to promoting creativity is **divergent thinking**: generating ideas by exploring many possible solutions. Often the most interesting ideas are **subversive**: they deliberately go against convention or obvious solutions.

You will **not** be judged on the quality of your art assets. It is fine to use assets found online, as long as they are available under an appropriate license and are properly attributed. Or it is perfectly acceptable to use placeholder "programmer art".

FAQ

- **What is the deadline for this assignment?**
  Falmouth University policy states that deadlines must only be specified

on the MyFalmouth system.
- **What should I do to seek help?**
  You can email your tutor for informal clarifications. For informal feedback, make a pull request on GitHub.
- **Is this a mistake?**
  If you have discovered an issue with the brief itself, the source files are available at:
  `https://github.com/Falmouth-Games-Academy/bsc-assignment-briefs`.
  Please make a pull request and comment accordingly.

Additional Resources

- `http://skilldrick.github.io/easy6502/`
- `http://nintendoage.com/forum/messageview.cfm?catid=22&threadid=7155`
- `http://www.obelisk.me.uk/6502/reference.html">6502instructionreference`

Marking Rubric

Criterion	Weight	Refer for Resubmission	Novice Competency	Novice Proficiency	Professional Competency	Professional Proficiency	Expert Competency
Basic Competency Threshold	40%	At least one part is missing or is unsatisfactory.	Submission is timely. Enough work is available to hold a meaningful discussion. Clear evidence of programming knowledge and communication skills. No breaches of academic integrity.				
Appropriateness of User Stories and Sprint Plans	5%	Few user stories are distinguishable and easily measured. Sprint plans provide little support for the project.	Some user stories are distinguishable and easily measured. Sprint plans provide some support for the project.	Most user stories are distinguishable and easily measured. User stories correspond to the product design. Sprint plans provide much support for the project.	Nearly all user stories are distinguishable and easily measured. User stories clearly correspond to the product design. Sprint plans provide considerable support for the project.	All user stories are distinguishable and easily measured. User stories clearly and comprehensively correspond to the product design. Sprint plans provide significant support for the project.	All user stories are distinguishable and easily measured. User stories clearly and comprehensively correspond to the product design. Sprint plans provide extensive support for the project.
Functional Coherence	10%	Few user stories have been implemented and/or the code fails to compile or run. Many obvious and serious bugs are detected. The game's performance is unacceptable.	Some user stories have been implemented. Some obvious bugs are detected. The game has many performance issues.	Many user stories have been implemented. There is some evidence of feature creep. Few obvious bugs are detected. The game has some performance issues.	Almost all user stories have been implemented. There is little evidence of feature creep. Some minor bugs are detected. The game has very few performance issues.	All user stories have been implemented. There is almost no evidence of feature creep. Some bugs, purely cosmetic and/or superficial in nature, are detected. The game has almost no performance issues.	All user stories have been implemented. There is no evidence of feature creep. Few to no bugs are detected. The game has no performance issues.
Sophistication	10%	Little insight into the appropriate use of programming constructs is evident from the source code. The program structure is poor or non-existant.	Some insight into the appropriate use of programming constructs is evident from the source code. The program structure is adequate.	Much insight into the appropriate use of programming constructs is evident from the source code. The program structure is appropriate.	Considerable insight into the appropriate use of programming constructs is evident from the source code. The program structure is effective. There is high cohesion and low coupling.	Significant insight into the appropriate use of programming constructs is evident from the source code. The program structure is very effective. There is high cohesion and low coupling.	Extensive insight into the appropriate use of programming constructs is evident from the source code. The program structure is extremely effective. There is very high cohesion and very low coupling.
Maintainability	15%	The code is only sporadically commented, if at all, or comments are unclear. Few identifier names are clear or inappropriate. Code formatting hinders readability.	The code is somewhat well commented. Some identifier names are descriptive and appropriate. An attempt has been made to adhere to a consistent formatting style. There is little obvious duplication of code or of literal values.	The code is reasonably well commented. Most identifier names are descriptive and appropriate. Most code adheres to a sensible formatting style. There is almost no obvious duplication of code or of literal values.	The code is reasonably well commented. Almost all identifier names are descriptive and appropriate. Almost all code adheres to a sensible formatting style. There is no obvious duplication of code or of literal values.	The code is very well commented. All identifier names are descriptive and appropriate. All code adheres to a sensible formatting style. There is no obvious duplication of code or of literal values.	The code is extremely well commented. All identifier names are descriptive and appropriate. All code adheres to a sensible formatting style. There is no duplication of code or of literal values.
Creative Flair	10%	Little or no creativity. The design is uncreative, showing no divergent and/or subversive thinking. The work delivers little or no fun and/or engagement.	Some creativity. The design demonstrates very little divergent and/or subversive thinking. The work delivers some fun and/or engagement.	Much creativity. The design demonstrates little divergent and/or subversive thinking. The work delivers much fun and/or engagement.	Considerable creativity. The design demonstrates some divergent and/or subversive thinking. The work delivers considerable fun and/or engagement.	Significant creativity. The design demonstrates significant divergent and/or subversive thinking. The work delivers significant fun and/or engagement.	Extensive creativity. The design demonstrates extensive divergent and/or subversive thinking. The work delivers extensive fun and/or engagement.

Criterion	Weight	Refer for Resubmission	Novice Competency	Novice Proficiency	Professional Competency	Professional Proficiency	Expert Competency
Portability and Navigability	5%	The game will not run at all. The code can only be compiled on the student's machine. The directory structure inside the submitted zip file is unclear.	The game runs on an NES emulator. The code can be compiled on another machine. The directory structure inside the submitted zip file is adequate.	The game runs on a variety of NES emulators, but with issues on some emulators. The code can be compiled on another machine. The directory structure inside the submitted zip file is mostly sensible.	The game runs on a variety of NES emulators with only minor issues. The code can be compiled on another machine. The directory structure inside the submitted zip file is sensible.	The game runs on a variety of NES emulators, and on NES hardware, with only minor issues. The code can be compiled on another machine. The directory structure inside the submitted zip file is sensible.	The game runs on a variety of NES emulators, and on NES hardware, with no issues. The code can be compiled on another machine. The directory structure inside the submitted zip file is sensible.
Use of Version Control	5%	Material has been checked into GitHub once per sprint or less.	Code has been checked into GitHub at least once per sprint.	Code has been checked into GitHub several times per sprint. Commit messages are clear, concise and relevant. There is some evidence of engagement with peers (e.g. code review).	Code has been checked into GitHub several times per sprint. Commit messages are clear, concise and relevant. There is much evidence of engagement with peers (e.g. code review).	Code has been checked into GitHub several times per sprint. Commit messages are clear, concise and relevant. There is significant evidence of engagement with peers (e.g. code review).	Code has been checked into GitHub several times per week. Commit messages are clear, concise and relevant. There is extensive evidence of engagement with peers (e.g. code review).