API HACKING

Version 1.0
BSc Computing for Games

Dr Edward Powley

Introduction

In this assignment, you are required to design and implement a component for a game. The component must be based on one or more **existing third-party APIs**: for example web services, hardware device SDKs, open-source libraries, or open dataset formats. Your component must be integrated into either:

(a) A game developed by **BA Digital Games** students (liaise with your chosen BA team before choosing this option); or
(b) Your **COMP130** Kivy game project; or
(c) Your **COMP150** group game project (**limit one per team**: liaise with the rest of your team before choosing this option); or
(d) An **open-source game**; or
(e) A **commercial game** which has a well-established (official or community-made) modding interface that allows mods to be developed in one of the permitted languages.

Whichever option you choose, you may implement your component in **C++**, **C#**, **Python**, or a combination of these. Other languages may be permitted at the discretion of your tutor, if you can argue convincingly that it is appropriate for your proposed project. Note that options (d) and (e) above are likely to be more technically challenging than the others, so are recommended only for those comfortable with advanced programming concepts.

You may work **individually** or **in pairs**. This project is assessed individually, so ensure that **your individual contribution is clearly indicated**.

This assignment is formed of **three** parts. There is a **single summative submission** via LearningSpace. Begin by forking the GitHub project at the following URL:

`https://github.com/Falmouth-Games-Academy/comp140-api-hacking`

A. *Propose* a game component

On GitHub, edit the `readme.md` file to contain a description of your proposed game component. Your proposal should:

- **Identify** the game into which your component will be integrated;
- **Describe** the component that will be created;
- **Show** the key user stories, in the form of a Trello task board;
- **Identify and justify** which technologies (implementation language, APIs, etc.) you will use.

Crypt of the NecroDancer uses the open-source *Essentia* library (`http://essentia.upf.edu/`) to analyse player-provided music tracks and synchronise gameplay with the beat.

Your proposal will be assessed on:

- **Scope**: is the proposed component non-trivial, but feasible given the duration of the project?
- **Appropriateness of component**: will the proposed component enhance the selected game?
- **Appropriateness of technologies**: is the choice of implementation language and APIs sensible and well-justified?

Formative submission: Discuss your proposal and task board with tutors in class.

B. *Implement* the component

You will build your prototype component over **three sprints**. At the beginning of each sprint, populate the sprint backlog on your Trello board. You should aim to have a 'potentially shippable' component at the end of each sprint; that is, a component which does not have any major flaws or half-finished features that prevent it from being tested. For the first two sprints, the component should be working either within the game or in a separate testing environment; for the final sprint, the component should be fully integrated into the game.

Your implementation will be assessed on the criteria listed in the marking rubric.

Formative submission: Participate in the sprint review sessions in class. Check your code into GitHub regularly (either your forked `comp140-api-hacking` repository or a branch of another appropriate repository), and make a pull request whenever you require assistance or feedback.

C. *Demonstrate* your component

Bring an executable of the game, with your component integrated, to the demo session in class. Be prepared to discuss it with tutors and peers.

This part is not assessed, but you will receive feedback which will be useful to you on future projects.

Formative submission: Participate in the demo session.

"There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies."

— C. A. R. Hoare

Summative submission (electronic)

Create a zip file containing the following:

- A markdown file named `readme.md` containing your **proposal** from part A, and any other **documentation** you feel is appropriate (e.g. special instructions for testing the component).
- Screenshots of your **Trello task board**. As a minimum, include one from the beginning and one from the end of each sprint, and screenshots documenting any checklists or other information within your cards.
- All **source code** created for part B. You do **not** need to provide the entire source code for the game. Any code which was **not written by you** must be **clearly identified** as such, in the `readme.md` file and/or as comments in the code files themselves.
- A **Windows executable** version of the game with your component integrated. Or, in the case of a mod for a commercial game, an executable version of the mod with clear installation instructions. **Do not include copyright material which you do not have permission to redistribute**.

ANGELINA, by Falmouth University researcher Michael Cook, uses fifteen different APIs and web services to automatically design entire games.

The recommended way to produce this zip file is to check all of the above into your GitHub repository throughout the course of the project, and then use the "Download Zip" function on the GitHub website. Any material you do not wish to upload to GitHub (e.g. the executable) must be added to the zip file manually before uploading.

Upload your zip file to the appropriate submission queue on LearningSpace. Note that LearningSpace accepts only a single zip file per submission, with a maximum file size of 1GB. If the game executable is larger than this, contact your tutors to make alternative arrangements.

Additional Guidance

Creating new software solutions from scratch is an important skill for software developers. However it can lead to "reinventing the wheel": spending much time and effort solving problems which have already been solved by others. Thus an equally important skill is being able to bring together existing code from different sources to produce a cohesive whole. Ensure that your **choice of technologies** is appropriate. Read up on the alternatives to determine which is the best fit for your proposed component.

Falmouth University is nationally and internationally renowned as an arts institution. Despite the fact that you are studying for a Bachelor of Science degree in a technical discipline, you are still expected to strive for the same level of **innovation and creative flair** as your fellow students in other departments. All assignments on this course involve a mix of technical and creative activities; cultivating both of these skills in tandem will stand you in good stead for a career in the games industry. One approach to promoting creativity is **divergent thinking**: generation of ideas by exploring many possible solutions. Often the most interesting ideas are **subversive**: they deliberately go against the conventional or most obvious solution.

The first step in planning your implementation should be to break your proposed component down into **user stories**. Your user stories should be **distinguishable** (i.e. there should be little overlap between them) and **easily measured** (i.e. it should be easy to tell when each user story has been implemented). They should also be **comprehensive**, i.e. the user stories should completely capture the desired functionality of the component, with no gaps. Your code will be assessed on **functional coherence**: how well the finished component corresponds to the user stories, and whether there are 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.

"Measuring programming progress by lines of code is like measuring aircraft building progress by weight."

— Bill Gates

Your code will also be assessed on **sophistication**. To succeed on a project of this size and complexity, you will need to make use of appropriate algorithms, data structures, library features, and object oriented programming concepts. Appropriateness to the task at hand is key, however: you will **not** receive credit for shoehorning a complicated solution into your program where a simpler one would have sufficed.

Maintainability is important in all programming projects, but doubly so when working in a team. Use **comments** liberally to improve comprehension of your code, and choose carefully the **names** for your files, classes, functions and variables. For high marks you should use a well-established commenting convention for **high-level documentation** of your files, classes and functions. The open-source tool Doxygen supports several such conventions. Also ensure that all code corresponds to a sensible and consistent **formatting style**: indentation, whitespace, placement of curly braces, etc. Hard-coded **literals** (numbers and strings) within the source should be avoided, with values instead defined as constants together in a single place.

Don't reinvent this.

Additional Resources

- M. G. Friberger et al. (2013) Data Games. Proceedings of Procedural Content Generation Workshop. FDG 2013. `http://julian.togelius.com/Friberger2013Data.pdf`
- Cook, M. and Colton, S. (2014) Ludus Ex Machina: Building a 3D game designer that competes alongside humans. Proceedings of ICCC 2014. `http://ccg.doc.gold.ac.uk/papers/cook_iccc2014.pdf`
- `http://docs.unity3d.com/Manual/NativePlugins.html`

Marking Rubric

Criterion	Weight	F (0 – 39)	D (40 – 49)	C (50 – 59)	B (60 – 69)	A (70 – 79)	A* (80 – 100)
Sprint reviews	Threshold 5% + 5%	None of the sprints are delivered, or no 'reasonable' peer reviews are submitted.		A 'potentially shippable' component is produced at the end of at least one sprint. A 'reasonable' review of at least one peer's work is provided in at least one of the review sessions.		A 'potentially shippable' component is produced at the end of all three sprints. A 'reasonable' review of at least one peer's work is provided in each of the review sessions.	
Choice of technologies (implementation language and APIs)	10%	The choice of technologies is inappropriate. Justification is inappropriate or not provided.	The choice of technologies is appropriate. Little appropriate justification is provided.	The choice of technologies is appropriate. Justification is appropriate, but no alternatives have been considered.	The choice of technologies is appropriate. Justification is appropriate, and alternatives have been considered.	The choice of technologies is highly appropriate. Justification is strong, showing a working knowledge of the chosen technology and its alternatives.	The choice of technologies is highly appropriate. Justification is exemplary, showing insight into the chosen technology and its alternatives.
Design of the solution	10%	User stories are not provided, or the design does not correspond to the user stories.	Few user stories are distinguishable and easily measured. The correspondence between design and user stories is tenuous.	Some user stories are distinguishable and easily measured. The design somewhat corresponds to the user stories.	Most user stories are distinguishable and easily measured. The design corresponds to the user stories.	Nearly all user stories are distinguishable and easily measured. The design clearly corresponds to the user stories.	All user stories are distinguishable and easily measured. The design clearly and comprehensively corresponds to the user stories.
Functional coherence	10%	No user stories have been implemented. There are critical bugs that prevent the code from compiling or running.	Few user stories have been implemented. There are many obvious and serious bugs.	Some user stories have been implemented. There are some obvious bugs.	Many user stories have been implemented. There is some evidence of feature creep. There are few obvious bugs.	Almost all user stories have been implemented. There is little evidence of feature creep. There are some minor bugs.	All user stories have been implemented. There is no evidence of feature creep. Bugs, if any, are purely cosmetic and/or superficial.
Sophistication	20%	No insight into the appropriate use of programming constructs is evident from the source code. APIs are used inappropriately. No attempt to structure the program is evident (e.g. one monolithic source file).	Little insight into the appropriate use of programming constructs is evident from the source code. APIs are used somewhat appropriately. The program structure is poor.	Some insight into the appropriate use of programming constructs is evident from the source code. APIs are used mostly appropriately. The program structure is adequate.	Much insight into the appropriate use of programming constructs is evident from the source code. APIs are used appropriately, with some mastery evident. The program structure is appropriate.	Significant insight into the appropriate use of programming constructs is evident from the source code. APIs are used appropriately, with much mastery evident. The program structure is effective. There is high cohesion and low coupling.	Exemplary insight into the appropriate use of programming constructs is evident from the source code. APIs are used appropriately, with exemplary mastery evident. The program structure is very effective. There is high cohesion and low coupling.
Maintainability	20%	There are no comments, or comments are misleading. Most variable names are unclear or inappropriate. Code formatting hinders readability.	The code is only sporadically commented, or comments are unclear. Some identifier names are unclear or inappropriate. Code formatting is inconsistent or does not aid readability.	The code is 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 consistent formatting style. There is almost no obvious duplication of code or of literal values.	The code is reasonably well commented, with some Doxygen-compatible module documentation. Almost all identifier names are descriptive and appropriate. Almost all code adheres to a consistent formatting style. There is no obvious duplication of code or of literal values.	The code is very well commented, with comprehensive Doxygen-compatible module documentation. All identifier names are descriptive and appropriate. All code adheres to a consistent formatting style. There is no obvious duplication of code or of literal values.

Criterion	Weight	F (0 – 39)	D (40 – 49)	C (50 – 59)	B (60 – 69)	A (70 – 79)	A* (80 – 100)
Innovation and creative flair	10%	Demonstrates no evidence of innovation and/or creativity. The solution is purely derivative of existing products. There is no evidence of divergent thinking.	Demonstrates evidence of emerging innovation and/or creativity. The solution is mostly derivative, with some attempts at innovation. There is evidence of an attempt at divergent thinking.	Demonstrates evidence of progressing innovation and/or creativity. The solution is an interesting and somewhat innovative product. There is some evidence of divergent thinking.	Demonstrates evidence of partial mastery of innovative and creative practice. The solution is a novel and innovative product. There is much evidence of divergent thinking.	Demonstrates some evidence of mastery of innovative and creative practice. The solution is a unique and innovative product. There is significant evidence of divergent thinking.	Demonstrates much evidence of mastery of innovative and creative practice.
Portability and navigability	5%	Game will not execute at all on another machine for reasons related to code portability which cannot be fixed easily due to its poor structure. The provided template has not been followed.	There were challenges executing the game, but these were resolvable. The directory structure inside the submitted zip file is unclear. The provided template has not been followed.	Several portability issues are present. The directory structure inside the submitted zip file is somewhat confusing. The provided template has mostly been followed.	Some portability issues are present. The directory structure inside the submitted zip file is adequate. The provided template has been followed.	Few portability issues are present. The directory structure inside the submitted zip file is mostly sensible. The provided template has been followed.	Almost no portability issues are present. The directory structure inside the submitted zip file is sensible. The provided template has been followed.
Use of version control	5%	GitHub has not been used.	Material has only been checked into GitHub a few times before the deadline.	Material has been checked into GitHub at least once per sprint.	Material has been checked into GitHub several times per sprint.	Material has been checked into GitHub several times per sprint. Commit messages are clear, concise and relevant.	Material has been checked into GitHub several times per sprint. Commit messages are clear, concise and relevant. There is evidence of engagement with peers (e.g. voluntary code review).