API HACKING

Version 1.0
BSc Computing for Games

Alcwyn Parker

Introduction

"For every complex problem there is an answer that is clear, simple, and wrong."

— Henry Mencken

"Bad programming is easy. (People) can learn it in 21 days, even if they are dummies...(Good programming requires a) willingness to devote a large portion of one's life to deliberative practice...So go ahead, buy that book; you'll probably get some use out of it. But you won't change your life or your real expertise as a programmer in 21 days...How about working hard to continually improve over 24 months? Well, now you're starting to get somewhere..."

— Peter Norvig

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 prototype should function as an add-on to either one of the games being developed by students on the BA Digital Games course; or the game that you developed in COMP150. Whichever option you choose, you may implement your component in C++, C#, Python, or a combination of these.

Application programming interfaces (API) allow programmers access to a vast array of third-party services and data-sources from social networks to real-time sensor networks. They are the conduits for communication between applications. Most software projects will involve API integration of some kind and so the focus of this assignment is to help you familiarise yourself with the concepts and techniques needed to integrate third-party services into your own projects.

This assignment is formed of several parts:

(A) **Write** a proposal for a game component that will:
  i. **state and justify** the game that will be the basis for your comonent;
  ii. **outline** the initial concept in detail;
  iii. **show** the key user stories, in the form of a Trello task board;
  iv. and **list** the key requirements the component must fulfil;
  v. including which technologies (implementation languages, APIs, etc.) you will use.

(B) **Implement** a draft version of your your game component :
  i. **improve** upon the design over the course of two weeks;

(C) **Implement** the final version of your game component that will:
  i. **revise** the design based on feedback from your peers;

(D) **Present** a practical demo of the game controller to your tutor that will:
  i. **demonstrate** your academic integrity;
  ii. as well as **show** your programming knowledge **and** communication skills.

Assignment Setup

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

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

Use the existing directory structure and, as required, extend this structure with sub-directories. Ensure that you maintain the `readme.md` file.

Modify the `.gitignore` to the defaults for **Python**. Please, also ensure that you add editor-specific files and folders to `.gitignore`.

A twitterbot sorting a picture by its pixels.

Part A

Part A consists of a **single formative submission**. This work will be assessed on a **threshold** basis. The following criteria are used to determine a pass or fail:

(a) Submission is timely;
(b) Design is feasible, distinctive and has creative merit.

To complete part A, on GitHub, edit the readme.md file to contain a description of your proposed game component.

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

Part B

Part B is a **continual formative assessment**. This work is collaborative and will be assessed on a threshold basis. The following criteria are used to determine a pass or fail:

(a) Version Control used effectively;
(b) Sufficient progress each week;
(c) Reflective practice.

You will build your prototype component over **three weeks**. You should aim to have a 'potentially shippable' component at the end of each week; 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 weeks, the component should be working either within the game or in a separate testing environment; for the final week, the component should be fully integrated into the game.

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

Formative submission: 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

Part C

To complete Part C, 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.

Part D

To complete part D, create a zip file containing the following:

(a) 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).
(b) Screenshots of your Trello task board to evidence your project management. and screenshots documenting any checklists or other information within your cards.
(c) 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.
(d) A Windows executable version of the game with your component inte-

grated. 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.

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.

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 D, prepare a practical demonstration of the game component. 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.

You will receive immediate **informal feedback** from your **tutor**.

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.

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.

FAQ

- **What is the deadline for this assignment?**
  Falmouth University policy states that deadlines must only be specified on LearningSpace. Please examine the assignment area where you located this document.
- **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 raise an issue and comment accordingly.

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

Criteria marked with a ‡ are shared by the group. All other criteria are individual.

Criterion	Weight	Refer for Resubmission	Basic Competency	Basic Proficiency	Novice Competency	Novice Proficiency	Professional Competency
Iterative development process	5%	There is little or no evidence of an iterative development process and no improvement over time in regards to the quality of the design and build of the prototype.		A ‘potentially shippable’ prototype is produced at the end of development period. There is evidence of a ‘reasonable’ iterative development process but the prototype suffers from ‘lock in’ in regards to the original concept.		A ‘potentially shippable’ prototype is produced at the end of the development period. The project has benefitted from an iterative development process and many improvement have been made to the original concept	
Choice of technologies (implementation language and APIs)	5%	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	15%	No user stories are provided, or the design does not correspond to the user stories.	Some user stories are distinguishable and easily measured. The correspondence between design and user stories is tenuous.	Little 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	5%	No user stories have been implemented. There are critical bugs that prevent the code from compiling or running.	Some user stories have been implemented. There are many obvious and serious bugs.	A little number of 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.	A considerable number of user stories have been implemented. There is little evidence of feature creep. There are some minor bugs.	A significant number of user stories have been implemented. There is no evidence of feature creep. Bugs, if any, are purely cosmetic and/or superficial.
Sophistication	10% ‡	No insight into the appropriate use of programming constructs is evident from the source code. No attempt to structure the program (e.g. one monolithic function).	Little insight into the appropriate use of programming constructs is evident from the source code. The program structure is poor.	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.
Maintainability	10% ‡	There are no comments in the source code, or comments are misleading. Most variable names are unclear or inappropriate. Code formatting hinders readability.	The source 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 source code is somewhat well commented. Some identifier names are descriptive and appropriate. An attempt has been made to adhere to thhe PEP-8 formatting style. There is little obvious duplication of code or of literal values.	The source code is reasonably well commented. Most identifier names are descriptive and appropriate. Most code adheres to the PEP-8 formatting style. There is almost no obvious duplication of code or of literal values.	The source code is reasonably well commented, with Python doc-strings. Almost all identifier names are descriptive and appropriate. Almost all code adheres to the PEP-8 formatting style. There is no obvious duplication of code or of literal values. Some literal values can be easily “tinkered” in the source code.	The source code is very well commented, with Python doc-strings. All identifier names are descriptive and appropriate. All source code adheres to the PEP-8 formatting style. There is no obvious duplication of code or of literal values. Most literal values are, where appropriate, easily “tinkered” outside of the source code.

Criterion	Weight	Refer for Resubmission	Basic Competency	Basic Proficiency	Novice Competency	Novice Proficiency	Professional Competency
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 some 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.	Source code has rarely been checked into GitHub.	Source code has been checked into GitHub at least once per week. Commit messages are present. There is evidence of engagement with peers (e.g. code review).	Source code has been checked into GitHub several times per week. Commit messages are clear, concise and relevant. There is evidence of somewhat meaningful engagement with peers (e.g. code review).	Source code has been checked into GitHub several times per week. Commit messages are clear, concise and relevant. There is evidence of meaningful engagement with peers (e.g. code review).	Source code has been checked into GitHub several times per week. Commit messages are clear, concise and relevant. There is evidence of effective engagement with peers (e.g. code review).
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. Clear evidence of reflection on own performance and contribution. Only constructive criticism of pair-programming partner is raised. No breaches of academic integrity.				