CAB202 Assignment 2, Exercise 1: Assignment 2

Return to Exercise List ›

Results so far:

No submission has been made so far. (0%).

Assessed weight: 0%. You may continue to attempt this item until you reach 100 submissions. So far, you have made 0 submissions.

Requirements:

CAB202 Assignment 2: Asteroid Apocalypse

Due Date:	03 June 2019 23:59:59
Submission:	Attach your solution using the file attachment controls provided below.
Assessment Weight:	40%
Revision Log:	The following changes have been made since the initial release of the assignment. • 07 May 2019 – Initial release of specification.

Introduction

In this assignment you will design, implement, and test, a simple interactive game called Asteroid Apocalypse which will run on the QUT TeensyPewPew device. Game functionality will be accessed via hardware controls connected to the Teensy, and also via character sequences received via USB serial connector.

Listen Up!

1. The assignment will be graded by strict reference to the criteria listed below.
2. **This is not a group assignment.** While we encourage you to discuss the assignment and brain-storm with your associates, you must ensure that your submission is your own individual work.

Share ideas, not code!

3. A high standard of academic integrity is required. Breaches of academic integrity, including plagiarism or any other action taken to subvert, circumvent, or distort the assessment process, will not be tolerated. QUT policy regarding academic conduct is available in the QUT MOPP Section C/5.3 Academic Integrity (http://www.mopp.qut.edu.au/C/C_05_03.jsp). In particular, under the provisions of MOPP statement C/5.3.7 (http://www.mopp.qut.edu.au/C/C_05_03.jsp#C_05_03.07.mdoc), Part (e), we reserve the right to require you to authenticate your learning. You may be required to show evidence of materials used in the production of the assignment such as notes, drafts, sketches or historical backups. You may also be required to undertake a viva or complete a supervised practical exercise.
4. Use of any third-party code library (other than the standard libraries supplied with GCC, ncurses, and the list of items labelled as Exceptions in the following paragraph) is strictly prohibited. Use of code downloaded from the internet, with or without correct attribution to the original author(s), is strictly prohibited. Submission of source code created by teaching staff of this unit in any previous semester is strictly prohibited. Subcontracting, outsourcing, off-shoring, purchasing, borrowing, stealing, copying, or obtaining source code by any means other than through an act of original creation by yourself, is strictly prohibited.

   Exceptions: you are strongly encouraged to call functions defined in the **current version** of the `cab202_teensy` library, as downloaded from CAB202 2019 Semester 1 Blackboard Learning Resources on or after 25 February 2019.

5. Do not post your solution in any form of online repository or file sharing service, or allow anybody else to obtain access to your solution in any way. Doing so will be classified as academic misconduct under the clauses pertaining to collusion, especially in the event that a copy of your source code obtained from such a service is submitted by another student, regardless of whether this has occurred without your knowledge and permission.
6. Abundant code samples, demonstrations, and exercises have been made available to support your effort toward this programming task. Written permission must be obtained from the Unit Coordinator if you want to use technology other than the ZDK to implement your game. Permission will only be granted if there are compelling special circumstances that make it impossible for you to use the ZDK. Without this permission, a game implemented with some other graphical framework will receive a score of 0. Direct use of the `ncurses` library to render graphics or text is expressly prohibited.

Breaching any of the foregoing conditions will result in an immediate and final score of 0 for the Assignment.

Deliverables

Use the controls at the bottom of this page to attach the following items:

- Implementation:
  - Submit all source files (`.c`) and header files (`.h`) required to compile and run your program.
- Test suite:
  - Submit a plain text document (`*.txt`) which fully and unambiguously defines your test

suite. This is a collection of concise, effective, and independent tests which will illustrate the capabilities of your program. The details of the contents of this script are set out under the heading Test Requirements, below.

Exciting introduction

In this game you control a starfighter which must defend the Earth from a catastrophic asteroid swarm. Asteroids enter the playfield at the top of the LCD display, and travel generally downwards. The starfighter is equipped with a potent plasma cannon mounted in a turret in its nose. When fired, this emits a coherent packet of highly energetic plasma (a plasma bolt) which travels in straight line from the launch point. The plasma cannon covers a 120-degree forward arc (i.e. in front of the starfighter), with default firing direction straight ahead. You can fire plasma bolts by aiming the turret at any point within the forward arc. When a plasma bolt hits an asteroid, the asteroid splits to form two boulders. Boulders continue to move downwards, at a new (different) oblique angle. When a plasma bolt strikes a boulder, the boulder splits to form two fragments, which in turn continue to move downwards at new oblique angles. The starfighter is protected by a deflector shield which extends horizontally across the display just in front of it. When the deflector shield is touched by an object (asteroid, boulder, or fragment), the object is immediately destroyed but the shield takes damage. Eventually, cumulative damage weakens the shield and the starfighter is destroyed by intense ionising radiation released as the shield collapses.

Functional Specification

1. **Controls:** The game will have controls on the Teensy and controls on the computer.

   Teensy Controls

   - Joystick left: move spaceship left
   - Joystick right: move spaceship right
   - Joystick up: fire plasma bolts
   - Joystick down: send and display game status
   - Joystick centre: pause game
   - Left button: start/restart game
   - Right button: quit
   - Left potentiometer: set the aim of the turret
   - Right potentiometer: set the speed of the game

   Keyboard controls

   - `'a'` – move spaceship left
   - `'d'` – move spaceship right
   - `'w'` – fire plasma bolts
   - `'s'` – send and display game status
   - `'r'` – start/reset game
   - `'p'` – pause game
   - `'q'` – quit

- `'t'` – set aim of the turret
- `'m'` – set the speed of the game
- `'l'` – set the remaining useful life of the deflector shield
- `'g'` – set the score
- `'?'` – print controls to computer screen (Putty)
- `'h'` – move spaceship to coordinate
- `'j'` – place asteroid at coordinate
- `'k'` – place boulder at coordinate
- `'l'` – place fragment at coordinate

2. **Deflector Shield:** There is a barrier across the screen in row 39. The barrier is depicted by a 1 pixel high dotted line.
3. **Starfighter:** There is a spaceship object.
    i. The Starfighter is at least 4 pixels and no more than 7 pixels in height.
    ii. The Starfighter is at least 5 pixels and no more than 15 pixels in width.
    iii. The Starfighter has a turret from which the linear wave-guide the Plasma Cannon extends. This will look like a short straight line and is included in the 7 pixel height limit. All further mentions of Starfighter include the turret.
    iv. When the game starts or resets, the Starfighter is horizontally centred in the bottom 8 rows (i.e. rows 40 - 47) of the screen, or as close to the centre as possible given the geometry of your design.
    v. When the game game starts or resets, the Starfighter is stationary.
    vi. No part of the Starfighter may ever leave the screen or overlap the Deflector Shield.
    vii. The Starfighter may never move vertically.
4. **Falling objects:** There are three types of falling object: Asteroids, Boulders, and Fragments.
    i. Asteroids fall from the top of the screen, two seconds after the game is started or restarted, and pose a danger to the Starfighter.
    ii. Initially, three Asteroids spawn at random horizontal positions above the top of the screen, and move smoothly into view one row of pixels at a time (i.e: they do not simply appear).
    iii. Asteroids may not go outside the left, right and bottom edges of the screen (but may be seen to come in from the top of the screen).
    iv. Asteroids fall downwards only.
    v. If a falling object touches the Deflector Shield, the object disappears and the remaining useful life of the Deflector Shield decreases.
    vi. When hit by a Plasma Bolt (see below) Asteroids split to form two Boulders, and Boulders split to form two fragments. Fragments just disappear when hit by a Plasma Bolt.
    vii. When an object splits, each of the two resulting objects acquires a new random velocity within += 30 degrees of the heading of the object that has been destroyed.
    viii. When a falling object is hit by a Plasma Bolt, the player's score increases.
    ix. Falling object sizes are as follows:
        - **Asteroid:** are 7 by 7 pixels, shaped like diamonds, and give 1 point when shot.
        - **Boulder:** are 5 by 5 pixels, shaped like diamonds, and give 2 points when shot.
        - **Fragment:** are 3 by 3 pixels, shaped like plus symbols, and give 4 points when shot.
5. **Game Status Information:** the game status is displayed in a clear and legible format and contains the following

On teensy screen

i. Game time
ii. Remaining Useful Life
iii. Score

Sent to computer

i. Game time
ii. Lives
iii. Score
iv. Number of asteroids on screen
v. Number of boulders on screen
vi. Number of fragments on screen
vii. Plasma bolts on screen
viii. Aim of turret
ix. Speed of game

6. **Game Status control:** the game status is controlled by pressing Joystick Down or `'s'` and causes game status information to be displayed the teensy screen or simply sent to the computer. If the game is not paused the game status information will be sent to the computer only. However, if the game is paused the 'On teensy screen' information specified above will be displayed on the screen along with the sending the status to the computer.
7. **Collision Detection:** may be bounding box or pixel level. However more marks will be awarded for pixel level collision detection.
8. **Starfighter Movement:** The Starfighter is controllable from the Teensy PewPew and a keyboard.
    i. At the beginning of the game, the Starfighter is randomly assigned a velocity, by default, that is speed 1 in either the left or right direction.
    ii. From a stationary position, pressing Joystick Left or `'a'` causes the Starfighter's velocity to increase in the left direction.
    iii. From a stationary position, pressing Joystick Right or `'d'` causes the Starfighter's velocity to increase in the right direction.
    iv. From a left velocity, pressing Joystick Left or `'a'` again, results in no change.
    v. From a left velocity, pressing Joystick Right or `'d'`, causes the Starfighter's velocity to change to 0, stopping the Starfighter's movement.
    vi. From a right velocity, pressing Joystick Right or `'d'` again, results in no change.
    vii. From a right velocity, pressing Joystick Left or `'a'`, causes the Starfighter's velocity to change to 0, stopping the Starfighter's movement.
    viii. When the Starfighter hits the side of the screen, it automatically bounces, reversing it's direction.
9. **Plasma Cannon Aiming Direction:** is controlled by the Left Potentiometer or `'o'` followed by numeric value.
    i. In the Plasma Cannon coordinate system, 0 degrees is taken to be straight up. The Plasma Cannon is restricted to aim between 60 degrees left of centre (-60) and 60 degrees right of centre (+60).
    ii. Setting the left potentiometer to its physical minimum position corresponds to -60. Setting

it to its maximum position corresponds to +60.

iii. The Plasma Cannon should visibly move to reflect the current angle as accurately as possible.

10. **Firing Plasma Bolts:** is controlled by pressing Joystick Up or `'w'`.
    i. Plasma bolts are fired if either the Joystick Up is pressed or `'w'` is received via USB serial, but only if at least 0.2 seconds has elapsed since the last Plasma Bolt was fired.
    ii. Plasma bolts travel in the direction at which the Plasma Cannon was aimed when they were launched, moving with constant velocity.
    iii. Plasma bolts disappear when they hit a falling object, or when they reach the edge of the screen.
    iv. The maximum number of plasma bolts on screen at any given time is 100.
    v. If Joystick Up or `'w'` is pressed and there are already 100 plasma bolts on screen, nothing happens.
11. **Introduction:** an introductory display appears on the Teensy when the game is initially loaded. The introduction can be exited/skipped by pressing Left Button or `'r'`. The introduction should include:
    1. Student number
    2. Game title
    3. Some kind of animation
    4. Use of PWM to adjust the brightness of the back-light (either on or off is up to you).
12. **Game initial setup:** the following occurs at the start of the game by default. Some items may change depending on cheats entered on the keyboard.
    i. The Starfighter and Deflector Shield are visible on screen.
    ii. The Remaining Useful Life of the Defelector Shield is 5.
    iii. The score is 0.
    iv. No Asteroids appear on screen.
    v. There are no Plasma Bolts on screen.
    vi. Game time is set to 00:00.
    vii. The game is paused.
13. **Game time and Game start:** the game time is derived from one of the Teensy's onboard timers and is incremented at approximately 1 second intervals. The game time starts when the player unpauses from game start. When the player unpauses the game status information is sent to the computer along with a "game started" message.
14. **Game pause:** the game is paused by pressing Joystick Center or `'p'`. When the game is paused the game time stops incrementing. To resume, Joystick Center or `'p'` is pressed again.
15. **Warning lights:** one of the LEDs flash at 2Hz before the Asteroids fall.
    i. If there are more Asteroids on the left side of the screen, only the Left LED flashes.
    ii. If there are more Asteroids on the right side of the screen, only the Right LED flashes.
    iii. The side the Asteroid is determined to be on is based on the middle of the Astroid.
16. **Game over:** when the players lives reach 0, the following happens:
    i. The current status is sent to the computer along with a "game over" message.
    ii. The backlight fades off using PWM.
    iii. The words game over are displayed on screen and both LEDs turn on for a duration of 2 seconds.

iv. The backlight fades back on using PWM, the LEDs turn off, and options to quit or restart are given.

17. **Restart game:** if restart is chosen, the game returns to it's Game start state.
18. **Quit game:** if quit is chosen, the Teensy screen goes into inverse mode and displays student number only.
19. **Speed of the game:**The Right Potentiometer or `'m'` is used to set the speed of the game. The speed ranges from 0 (no movement) to a suitable speed that is still playable.
20. **Direct screen write:**additional marks will be awarded for drawing and updating the Starfighter using direct screen write. If direct screen write is used, only the minimum screen area necessary should be updated via this method with the rest of the screen being updated normally via the cab202_teensy screen buffer.
21. **Game cheats:** Allow the player to enter commands into the computer console to change the state of the Teensy Asteroid Apocalype game.
    i. `'l'` - set the lives
    ii. `'g'` - set the score
    iii. `'?'` - print controls to computer screen (Putty)
    iv. `'h'` - move spaceship to coordinate within the bottom 8 rows
    v. `'j'` - place large asteriod at coordinate
    vi. `'k'` - place medium asteriod at coordinate
    vii. `'l'` - place small asteroid at coordinate

Program structure, implementation quality

As is always the case, the program must be implemented with a view to ease of comprehension and maintainability. To this end:

1. The program should be subdivided into loosely coupled functions. Wherever possible and practical, data is transferred between functions via parameters and return values rather than shared global variables.
2. No function may contain more than 25 executable C language instructions. Note that comma-separated expressions with side effects will be classified as multiple executable instructions.
3. No function may be substantially identical to any other function.
4. All functions have carefully and meaningfully narrated documentation comments.
5. All source code must be formatted in a professional manner, making sensible use of meaningful identifiers, and adhering to a consistent coding standard of your choice.
6. Groups of related functions should be sectioned out into separate modules. Use of global variables is restricted to the compilation unit within which it is declared.

Test Requirements

Provide a test suite which defines a comprehensive but concise list of test definitions which rigorously and methodically cover all items of functionality.

Each test must be documented in the shell script, using in-line comments to address:

- What item of functionality (from the numbered list above) the test exercises.

- What the expected outcome from running the test should be.
- What the behaviour observed in your program is.
- An executable command which includes the explicit input sequence required to run the test, and which executes the test automatically when the shell script runs.

A template text document will be published as a tutorial resource during Week 11. Failure to provide a correctly will result in a score of 0 for this section.

Marking

The assignment is worth 40% of your total grade in this subject, and it is marked out of 30. A detailed marking rubric will be published in Blackboard Grade Centre at least 14 days before the submission deadline. The approximate breakdown of marks will be:

- Functionality: 25 marks.
- Testing: 10 marks.
- Program structure, implementation quality: 5 marks.

The following points should be noted about marking:

1. **If your code does not compile when submitted to AMS, the mark awarded will be 0.**
    - Give yourself plenty of time to get the basics right.
    - Submit Early. Submit Often.
2. **If the program has been implemented via a framework other than the CAB202 Teensy library without prior written permission from the Unit Coordinator, the mark awarded will be 0.**
3. If the program locks up or produces other untoward brick-like behaviour, marks will be awarded for the features that were observed prior to the point at which the game ceased to operate.
    - No effort will be made to work around the crash, nor will we debug your code to make it compile or run.
    - "It runs fine on my computer!" may be true, but it is largely irrelevant if the program crashes when executed on another machine. Such an excuse will not be accepted.
4. It is your responsibility to implement each feature sufficiently well that it is readily detected through normal operation of the game. Any feature that is not apparent through normal play will be deemed to be unimplemented.
5. We require tutors to adhere to a strict time limit of 3 minutes run time when marking each submission. Any feature that cannot be assessed in that time will be deemed to be unimplemented. Therefore you must avoid defects in game dynamics such as extremely fast motion, extremely slow motion, inconsistent control settings, premature program termination, or other properties that render the game unfit for use.
6. It is better to implement some of the features extremely well than to try to do everything and deliver a substandard product.
7. Penalties will be applied if the code exhibits general defects or undesirable behaviour not otherwise covered in this document. This includes but is not limited to such things as:
    - Errors in object motion, such as objects jumping more than one character position per frame;
    - Objects moving outside their permitted area;

- Failure to clear the screen appropriately between updates;
- Collisions involving invisible or non-existent objects;
- Disturbing or flashing display;
- Sluggish response times, excessively fast response times, or other performance defects which render the simulation difficult to operate according to specification.
- Gratuitous errors in program structure and organisation, including but not limited to: inappropriate use of recursion; use of inappropriate data structures such as linked lists or binary search trees; using `#include` to insert the contents of source files rather than header files; uploading of multiple versions of the same source file.

Notes:

- For purposes of AMS assessment, this activity has been classified as non-assessed. This does not mean the assignment is non-assessable – it means that the system is not able to assess the assignment automatically. AMS will not enforce a hard close-down for submissions, however in line with QUT policy, late submissions without an approved extension will not be marked. If special circumstances prevent you from meeting the assessment due date, you can apply for an extension (https://www.student.qut.edu.au/studying/assessment/late-assignments-and-extensions) *before the due date*. If you don't have an approved extension you should submit the work you have completed by the due date and it will be marked against the assessment criteria.
- If your solution consists of a single C source file, you may either paste the contents into the text area provided below, or use the "Attach file" button to load your file into a new text area.
- If your solution consists of multiple C source files and header files, as indicated by a suffix of `*.c` or `*.h`, use "Attach file" once for each file.
- A maximum of 30 files may be included in each submission. This consists of the built-in "submission.c" document, plus 29 attachments.
- Attach only one version of your program in any single submission.
- When you have attached all required files, press the "Submit" button.
- Source files for each submission will be placed in a single distinct folder on the server, and compiled with the following command:

```
C:/WinAVR-20100110/bin/avr-gcc \
        *.c \
        -mmcu=atmega32u4 \
        -Os \
        -DF_CPU=8000000UL \
        -std=gnu99 \
        -I../cab202_teensy \
        -L../cab202_teensy \
        -Wl,-u,vfprintf \
        -lprintf_flt \
        -lcab202_teensy \
        -lm \
        -o a2_n9999999.obj
C:/WinAVR-20100110/bin/avr-objcopy -O ihex a2_n9755373.obj a2_n9999999.hex
```

where n9999999 represents your student number.

Therefore, you should organise your files in a folder alongside the `cab202_teensy` folder and use this command to build your solution. Make sure there are no additional files (such as old versions of your program) in your work folder to prevent unpredictable build errors due to inclusion of incorrect versions of files.

- If compilation is successful, AMS will verify that your program has compiled successfully and return. Your program will *not* be executed because there is no meaningful test that can be performed automatically on a program such as this.
- The transcript will contain a copy of the compiled `.hex` file. You can paste this into a text document (using notepad++ or a Linux-compatible text editor) and save it as a `*.hex` file for local testing.

Submitted files:

Attach file(s):
选取文件 未选择文件

Use the file chooser to attach a source file.

Declaration and submission

By submitting this form, I certify that:

- I have read, and understand, QUT Manual Of Policy and Procedures, Section C/5.3, Academic Integrity; and
- This submission is in full compliance with all provisions of QUT Manual of Policy and Procedures, Section C/5.3, Academic Integrity; and
- With the exception of support libraries provided to the class by the CAB202 teaching team, I am the sole author of all source code and attachments included in this submission.

Agree to these conditions: ☐

Submit

Transcript: