CS1830/CS1831 – 2017/2018

Group Project

Due date:	March 16th at 23:55
Submission method:	Moodle submission
Feedback:	April 19th
Weight:	40% of your final mark

Learning outcomes assessed

In this assignment you will implement a 2D action game in Python using the `simplegui` library. The assignment covers all of the Python programming techniques from the course, including collision detection between moving sprites.

Instructions

The groups for this project will be published on Moodle at the end of the 7th week of the term. Your work will be submitted electronically on Moodle as well. Read the instructions there.

You will submit a ZIP file with the following contents:

1. `readme.txt` with brief instructions on how to start your project (namely the Python version used); if you have developed in CodeSkulptor, the link to your code should also be in the file;
2. a file `myGame.py` or a folder `myGame` (if your project has several files) with the contents of the project; if you have included a CodeSkulptor link in the file `readme.txt`, this code should match the code available at the link;
3. an image or PDF file with a Statement of Relative Contribution signed by all the group members.

Marking criteria

A detailed marking grid is presented throughout this document. Group feedback will be provided.

All the work you submit should be solely your own work.
Coursework submissions are routinely checked for this.
Failure to comply will result in sanctions.

1 Introduction

You will build a 2D action game using the `simplegui` library, with material from lectures and lab sessions. The set of mandatory features listed in this document will allow you to have 60% of the full credit. The other 40% will be attributed to additional features (some suggestions are provided).

2 Mandatory features (60%)

2.1 Game characteristics [5% of the grade]

– The program implements an action game where the player controls a sprite (e.g., a character) that fights against the computer.
– The player has a limited number of lives (three is a good number).
– There is a realtime score mechanism.

2.2 User interface [5% of the grade]

– The program displays a welcome screen when it starts.
– When the lives go to zero, the welcome screen reappears and all the game sprites are cleared.
– While the welcome screen is being displayed the game mechanism is stopped.
– When the welcome screen is clicked, the lives and the score are reset.
– The program displays on the canvas an appropriate text for both the lives and the score.

2.3 Player sprite [10% of the grade]

– The player controls an animated sprite.
– The sprite animation depends on the control (e.g., a character can me moving to the left or to the right, and the animation should reflect this).

2.4 Vectors [10% of the grade]

– The programme uses the `Vector` class (developed during lectures and lab sessions) to position each sprite on the canvas.
– Some of the game sprites are subject to a velocity vector and move accordingly on the canvas.

2.5 Collisions [15% of the grade]

– The game includes collisions between the sprite controlled by the player and other sprites (e.g., to check if a projectile hits a target).
– The collisions are handled using the techniques described in the lectures.

2.6 Object-oriented approach [15% of the grade]

– The program follows an object-oriented approach.

– The sprite controlled by the player should be wrapped inside a `Player` class.
– Other game sprites should be wrapped inside appropriate classes.
– The collisions between the sprite controlled by the player and other sprites should be wrapped inside an `Interaction` class.
– The main aspects of the game (state variables, or constants for canvas dimensions, images, etc.) should be wrapped inside a `Game` class. This class can also encapsulate the game loop.

3 Additional features (40%)

To qualify for the other 40% of the credit, your game must support additional features such as those included in the list below. The amount of credit to be granted will be determined by the course staff based on the perceived effort invested into implementing these features.

– Animated backgrounds or other interesting animation effects (explosions, etc).
– Two-player cooperative or competitive game.
– Sprites not controlled by the player colliding between them, following physics principles.
– Some of the sprites move with acceleration.
– Interesting story lines (e.g., multiple levels of increased difficulties).
– Pursue and evade effects (e.g., sprites that “gravitate” towards or chase the player).
– Other nontrivial features that enhance the gaming experience.

4 Recommendations

This is a group project: make sure that all the team members actively participate in the design and development of the game. Also, even if some code snippets will probably be attributable to one author, each team member should clearly understand the overall structure of the program.

If you know how to work with a versioning system, like for instance Git, you can use it to maintain a repository for the project. Alternatively, you can share CodeSkulptor links with the other team members. Whatever solution you adopt, make sure that you keep the code synchronised across the team.

When designing the game, make sure that you cover the mandatory features before you move on to the additional material. Also, never loose track of the playability and coolness of your game.

5 Statement of relative contribution

This statement reflects the agreed relative contributions of the group members (the percentage of work done by each one), which should add to 100%. The statement must be signed by all team members and scanned. These percentages will be used to adjust the final grades of the Group Project. We recommend that you recognise the effort and time spent by each member of the group, even if not all of the results of that effort were fully usable.

HAVE FUN