Instruction for project; project number 2

Basics of Computer Programming 2020/21, Informatics and Data Engineering

author: Robert Ostrowski[1]
version: 25.11.2021 r.

Project Bullet Hell

Goal

Goal of the project is implementation of a game from a bullet hell genre and survival (of the semester). The game involves looking at a character from a top-down perspective, dodging a lot of bullets and shooting at the enemy. Specific elements and functionalities of the game required to be implemented are given below.

Examples of games from the genre:
https://www.youtube.com/watch?v=zmyDrJJaLoo&t=617s TouHou
https://www.youtube.com/watch?v=55qSLBjDWHA Enter The Gungeon
https://www.youtube.com/watch?v=mX-wPxFsPgQ Undertale combat

Theme of the game is left up to the student, as long as there is a significant number of "bullets" on the screen at once.

Programming environment

To the instruction there is attached a basic program which includes:
- Calculation of time increases, which allows to measure the time spent
- Drawing bmp graphic files
- Drawing a pixel, a line, and a rectangle
- Displaying a text

The program uses SDL2 (2.0.3) – http://www.libsdl.org/ library. It is included in the basic project – there is no need to download the sources.
The compilation under a 32-bit Linux system can be done be the following command:

```
g++ -O2 -I./sdl/include -L. -o main main.cpp -lm -lSDL2 -lpthread -ldl -lrt
```

and under a 64-bit system with the following command:

```
g++ -O2 -I./sdl/include -L. -o main main.cpp -lm -lSDL2-64 -lpthread -ldl -lrt
```

1 In the case of doubts or inconsistencies in the instruction please contact the author. The office hours are specified at enauczanie.

To compile the project, the directory containing main.cpp should contain:

- Bitmaps with the needed pictures (cs8x8.bmp, eti.bmp), be sure to preserve proper capitalization of letters.
- The libSDL2.a file (libSDL2-64.a when compiling a 64-bit version)
- The sdl directory attached to the project.

To the project, there are attached scripts that can be used for a compilation (comp for a 32-bit version environment and comp64 for a 64-bit version environment).

The presentation (for the grading) of the project will be in the operating system chosen by a student. The following choices are available:
Linux system: The student is required to check if the project can be properly compiled and runs correctly under the laboratory distribution.
Windows System and compiled using MS Visual C++ studio with the version available at the laboratory.

The successful execution of the program during the presentation is required to obtain points from Project 2.

The program shall not use the C++ stl library.

Obligatory requirements (5 pt.)

All elements enumerated below are required. Lack of any of the following elements results in receiving 0 points for this project.

1. Preparation of graphics for the game: stage view, a place for displaying additional information: time elapsed from the start of the level.

2. Implementation of control using the following keys:
   a. *Esc*: exit the program - the game shuts down immediately;
   b. *n*: new game;
   c. arrows - movement

3. Implementation of one level of the game. The stage should be at least several times wider and higher than the displayed view. Traversal should correctly display the visible part of the stage.

4. There is no need to implement any end condition, but the elapsed time for a level should be measured properly

5. Implementation of one, static enemy shooting in some simple pattern.

Extra functionalities (10 pt)

1. **(1 pt)** Advanced enemies:
    a. at least 3 types of enemies with different shooting patterns made of simple bullets
        i. bullets should form a shape on the screen e.g. circles expanding in waves or a spiral
        ii. a shooting pattern of an enemy should not be static through the whole level
        iii. one type of an enemy should be able to move
    b. at least 3 stages which demonstrate different configurations (one enemy per stage)
    c. **attention:** this point will be expanded in point 8

2. **(1 pt)** Advanced attack patterns of enemies
    a. highlight a shape on the ground - after a brief delay a character still remaining in the area is hit (2 different shapes)
    b. shoot at a place, when the projectile arrives it explodes, creating a circular pattern of bullets
    c. shoot a single projectile of a significantly greater size
    d. these attacks should have a limited frequency and be visually distinct

3. **(1 pt)** Shootout:
    a. the player character can fight back and shoot as well
    b. implement hitboxes for both sides of the conflict
    c. after being hit the player cannot be hit again for a brief period of time (invincibility frames)
    d. it is sufficient to track number of hits for each party as a placeholder number (see other points for further development)

4. **(1 pt)** Health, death and menu:
    a. health should be displayed in a graphic format (e.g. bar)
    b. implement a menu which allows the player to navigate through the implemented options. At least the following should be available:
        i. start a new game on a given level
        ii. quit game
    c. depleting health of the player should display a game over and ask for another try or lead back to menu
    d. depleting health of the enemy should prompt the player to access the next level (ask to save the score if implemented)

5. **(1 pt)** Keep track of the score:
    - **a.** the score should increase when the enemy is hit and deplete when the player character is hit
        - i. multiple hits in a row should increase the magnitude of change
    - **b.** bonus points:
        - i. find and collect bonus health appearing randomly in the level
    - **c.** the score should be reflected in form of a 'grade' for a player, which should be displayed prominently and proudly

6. **(2 pt.)** Animations (of the implemented functionality):
    - **a.** animate the movement of player character
    - **b.** animate the movement of enemy character
    - **c.** animate the destruction of relevant subjects (if implemented)
    - **d.** changing the grade should trigger a relevant animation (if implemented)
    - **e.** **attention:** assuming that the minimal requirements for the game are fulfilled the speed of animations should not depend on the speed of the computer! E.g. you can define speed and keep track of changing positions using time.

7. **(1 pt)** Record high scores (number of hits if counting is not implemented):
    - **a.** after the level ends the game should ask the player to input their nickname and save their score in a file
    - **b.** sorted list of high scores should be accessible from menu
    - **c.** the number of positions visible on the screen at once should be limited
    - **d.** if not every score can be displayed at once it should be possible access the other (for example by scrolling or changing pages)

8. **(2 pt)** Encode a level in a file

    - **a.** An individual file format to store information about a level shall be developed. The files should be editable – for example in a notepad. The student shall be familiar with the format enough to discuss it and edit a level during the presentation. The program shall be able to load the description of a level from the file.
    - **b.** The file shall contain the information about the position of all of the elements

and the size of level, at least.

c. In addition, the file shall contain the information about all extra functionalities implemented and allow to mix and match them
   i. in particular the file should allow the user to customize the shooting pattern of an enemy and not be restricted to those in point 1.
   ii. if an element is random the file should specify some of its parameters, e.g. frequency
d. **Attention:** The program shall not limit the maximum number of objects in a file. It means that the program shall be able to analyze the content of the file and allocate the memory to store the representation of all the objects contained in the file. The encoding of this information shall be developed individually. It means, in particular, that a kind of preamble can be added to the file, to describe the objects contained in the level, and the size of the level. This allows the program to read the preamble, allocate memory according to the preamble, and then to load a level. The consistency of the preamble and the data shall be checked.

Bonus requirements (3 pt.)

Attention: the requirements below are graded only if all other points have been implemented.

9. **(1.5 pt) missiles**
   a. a distinct type of projectile that follows a curved trajectory- for example defined by Besier curves of given parameters
   b. it should be possible to encode the trajectory in the customization file and create a pattern made of missiles
10. **(1.5 pt) custom hitboxes**
   a. the customization file should specify the shapes of elements of the game: player, enemy and special attacks *a* and *c* from point 2
   b. if no suitable texture is provided a geometrical shape should be displayed
   c. the collision detection should work for the specified shapes

Final observations:

- The same code should be presented as submitted to the STOS system.
- Requirements about graphics: it is enough to use any bitmaps (of a proper size).
- The configuration of the program shall allow to change all the parameters easily, not only those considered in this instruction. The easy change means changing a parameter in the code.
- The project can be written using objects, but using standard C++ library (in particular string class, cin, cout, vectors, etc.)is forbidden . You can use the

content of string.h library from C std language. Do not confuse the content of this library with operations on string type in C++.

- The files shall be processed using standard C library – the family of f* functions (like fopen, fread, fclose, etc.).
- Every part of the code submitted for the grading should be implemented individually by the student.
- The speed of the game shall be independent of the speed of the computer on which the program is run.
- The constant parameter in the program shall be described using suitable comments, for example:

```
const int WIDTH_OF_BOARD = 320;    // pixels
const double XPOSITION_OF_TEXT = 60.0; // % of the screen width
const double YPOSITION_OF_TEXT = 5.0;  // % of the screen height
```