Instruction for project 2

Basics of Computer Programming 2023/24, Data Engineering

author: Robert Ostrowski[1]
date: 29/10/2023

King Donkey Project

The goal

The goal of the project is to implement the "King Donkey" game and avoid a lawsuit for similarity with https://www.youtube.com/watch?v=Pp2aMs38ERY. The game is about controlling the movement and jumps of a character traversing a devastated building and avoiding barrels on the way to the top (for points for the project). Selected functionalities/game elements that have to be implemented are listed below.

Detailed game description can be found on the following page:
https://en.wikipedia.org/wiki/Donkey_Kong_(arcade_game)

Programming Environment

The instruction comes with a helper program that implements:
- time increment calculation, which allows you to track its passage;
- displaying graphic files in the BMP format on the screen;
- drawing pixels, lines and rectangles;
- text display.

The program utilizes the SDL2 library (2.0.3) – http://www.libsdl.org/. It is included in the helper program and there is no need to download its sources.

Compilation under Linux is performed using the following command (in a 32-bit system):

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

or (in a 64-bit system):

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

In order to successfully compile the helper program, the main.cpp file should be in the directory with

[1] Note: In case of any ambiguity in the instruction, please contact the author; consultations take place on Wednesdays 6-8 p.m., more information can be found in the "Basics of Computer Programming 2023/2024" course on enauczanie.

- bitmaps with required drawings (cs8x8.bmp, eti.bmp);
- libSDL2.a file (libSDL2-64.a for 64-bit compilation);
- sdl directory attached to the program.

The program includes scripts that can be used for compilation (comp in a 32-bit environment and comp64 in a 64-bit environment).

The presentation of the program (required if you want to get any points) will take place in an environment chosen by you from the following two options:

- Linux: you should check, before coming to the examination, whether the program compiles and runs correctly under the distribution available in the laboratory;
- Windows: in the MS Visual C++ environment in a version compatible with the one available in the laboratory.

Showing that your program runs (during the presentation) is a necessary condition for obtaining any points.

The C++ **stl** library should not be used in your program.

Mandatory Requirements (5 points)

All items listed here must be implemented. The version of basic requirements is very simplified, in particular the control can be used to test further requirements. The absence of any of the following elements results in no points.

1. Preparing the graphic design of the game: outline of the board, preparing a place to display additional information (e.g. time that has passed since the beginning of the stage). Control key implementation:
   a. ***Esc***: exit from the program - the program ends immediately;
   b. ***'n'***: new game.

2. Implementation of one stage of the game. The stage should have a floor and several horizontal, hanging platforms connected by ladders. The scene size can be limited to the window size.

3. Implementation of the mechanics of moving the character using arrow keys. Left and right movement should be limited by platforms, and up and down by ladders. The position of the character should be limited by the resolution of the game, not the position of the blocks from which the scene is built.

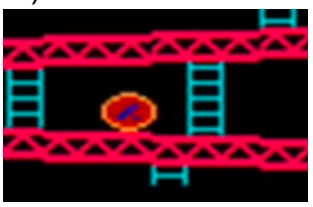
4. There is no need to implement the ending condition (fail or exit), but the elapsed time should be measured in a proper way. The time should be displayed along with the information about implemented requirements (mandatory and optional).

Optional Requirements (10 points)

A. **(1 pt)** Implementation of the jumps:
   a. After pressing the **spacebar**, the character should jump. Jump height should be limited. It should be possible to jump over the barrels (if point C is implemented) and also jump onto the platform. After going beyond the edge of the platform, the character should fall until it hits the platform or the ground below.
   b. Character movement and control parameters should be easy to change to achieve smooth control.

B. **(1 pt)** Implementation of more than one stage:
   a. At least 3 different stages should be available.
   b. To facilitate demonstration, the stage is changed by pressing the corresponding numeric key.
   c. The program should detect entering the area ending the stage and signal this event (continued in point D).

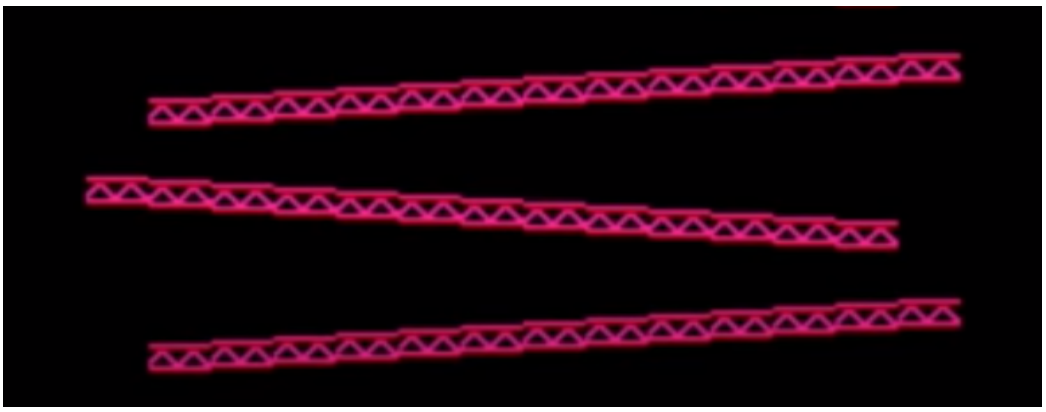
C. **(1 pt)** Barrels
   a. The barrels move starting at a specific location and traveling along a fixed route to the end point or following simple rules.
   b. The behavior of the barrels should be parameterized.
   c. The program should detect a collision with the barrel and signal this event (continued in point D).



D. **(1 pt)** Death, number of lives and the menu:
   a. After starting the game, a menu should appear allowing the player to select all options: exit, save, read, check results, select a stage. Selecting an option that is not implemented should display a message stating that it is unavailable.
   b. The text entered by the player (e.g. during writing a nickname or the name of the save file) should be always visible and the ***backspace*** key should allow removing letters.
   c. Displaying the remaining number of character's lives on the screen in graphical form.
   d. Loss of life when touching a barrel (if point C is implemented).
   e. Losing a life should prompt the player to continue and show (if point F is implemented) the number of points the player has gained.
   f. Losing all lives should bring up the main menu after asking whether to save the result (if point H is implemented).
   g. After reaching the top, proceed to the next stage. (It is suggested to prepare a simple way to demonstrate this functionality in advance.)

E. **(2 pts)** Animations (requires at least 2 of the 3 additional items described below):
    a. Animations of running, jumping (point A) and climbing of the character.
    b. Animations of the antagonist (i.e. the source of the barrels) responsible for dropping the barrels (i.e. signaling that the barrel is about to appear).
    c. Animations of the barrels (point C*)*.
    d. The bonus points you have earned (point F) should appear on the screen for a moment.
    e. **Note:** animation speed should not depend on the speed of your computer (assuming it meets the minimum requirements of the game)!
F. **(1 pt)** Counting:
    a. The number of points should be displayed on the screen and dynamically changed.
    b. Completing a stage increases a player's points. They should be kept when moving to the next stage (or restarting the current one if point B is not implemented).
    c. Jumping over barrels (if point C is implemented*)* results in extra points.
    d. Placing a trophy that gives you extra points when touched at a specific point in a stage.

G. **(1 pt)** Saving best scores:
    a. Once the player finishes playing, he should be able to enter his nickname and score into the file. The number of results saved in the file should not be limited.
    b. From the menu you can view a sorted list of results.
    c. The number of results on the screen, as opposed to those contained in the file, should be limited.
    d. If not all results fit on the screen, the program should allow the player to reach them, e.g. by switching between pages or scrolling through the list.
    e. If the F point is not implemented, time can be recorded instead of results.
H. **(1 pt)** Advanced game stage look:
    a. The hanging platforms can be tilted (or convincingly pretend to be tilted).



I. **(1 pt)** Encoding the appearance of a game stage in a file:
   a. You should design your own (editable, e.g. in a text editor) game stage file format. This format should be familiar to you so that you are able to explain and make the indicated step edits during the presentation. When starting a given stage, the appearance of this stage should be read from the file. The file should contain at least information about the location of all obstacles in this stage and the dimensions of the stage.
   b. Additionally, the file should contain information about all elements with which the implementation was extended when implementing optional requirements.
   c. **Note**: the program should not impose limits on the maximum number of objects of various types contained in the file. This means that the program analyzes the contents of the file and then allocates enough memory to store data about all the objects in the file. The format for encoding this information in the file should be selected by you, which means that to make reading the stage description from the file easier, you can choose a format in which at the beginning of the file there is information (preamble) about the number of individual objects, and then there is the board description. This way you can load the preamble, allocate memory based on the data there, and then load the stage. (With this solution, data validation should be added - the program should recognize any inconsistency between the preamble and the content)

Extra Requirements (3 points)

Note: the following requirements extend the subpoints: game stage appearance, collisions, game stage appearance coding and are only assessed when all other points have been implemented.

J. **(1 pt)** Physics
   a. Platforms can be at any angle and are modeled as inclined planes.
   b. The barrels should move and collide according to specific rules of in-game "physics", rather than a predetermined trajectory. The criterion

for disappearance can be a duration limited by a parameter.

K. **(2 pts)** Dynamic stage creation – special stage in infinite mode
   a. The generation rules should be parameterized.
   b. When the player climbs above half of the screen, another part of the stage should be generated. The camera follows the player upwards, revealing the generated scene.
   c. Platforms and ladders should be generated in such a way that the player can always climb higher.
   d. Reaching higher and higher designated places in the stage should be scored.
   e. The antagonist (i.e. the source of the barrels) should also change position when reaching such a location.

Final Remarks

Graphic design requirements: it is sufficient to use any bitmaps (appropriately selected in terms of size).

The program configuration should enable easy change of all parameters, not only those clearly indicated in the above description. By easy change we mean modifying a constant in the program.

The program can be written in an object-oriented way, but it is completely forbidden to use the C++ standard library (including the types string, cin, cout, vector, etc.) (Note: the string type from the C++ library should not be confused with the string.h library from C – you can use functions located in string.h)

File handling should be implemented using the C standard library (f???? family of functions - e.g. fopen, fread, fclose, etc.)

Each fragment of the code submitted for assessment should be written independently by you.

The speed of the program should be independent of the computer on which the program is running. Constant units in the program should be described with appropriate comments, for example:

```
const int BOARD_WIDTH = 320;    // pixels
const double TEXT_X_COORDINATE = 60.0; // percentage of the width of the screen
const double TEXT_Y_COORDINATE = 5.0;  // percentage of the height of the screen
```