U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Totally Not Mario 2D!

LCOM

Class 8 – Group 1

André de Jesus Fernandes Flores – up201907001

Diogo Luís Araújo de Faria – up201907014

Contents

User Instructions

To compile the game only the make command is needed. To run the game, you need to specify the file path of a top score file, one is already included in the same directory of the makefile.

Ex: lcom_run proj "/home/lcom/labs/proj/top_score.txt"

Drag the mouse and left click on the options:

\- Starts the game

\- Quits the game

In the bottom right corner of the screen, in any menu, the real time, according to the RTC, is displayed.

If the game is started by left clicking on the start box, the game begins:

To move the character, you must use the keyboard:

- A – to move to the left
- D – to move to the right
- Space bar – to jump

 - This is an enemy. By jumping on top of it, it is killed. By impacting it any other way, the player loses one of their lives.

 - This is a coin. By colliding with it, the player score is incremented by 1 and the coin darkens.

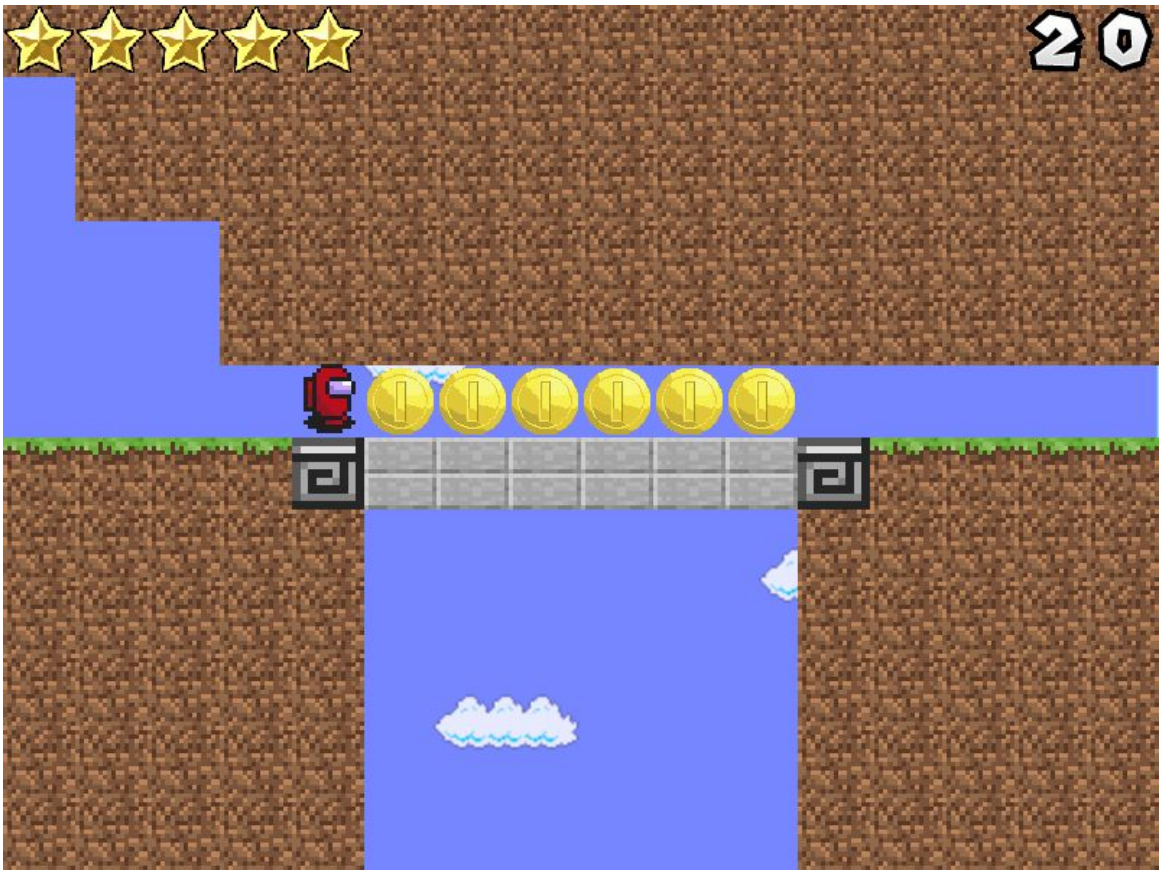
The character must reach the right most point of the screen and walk through the threshold, which makes the next game level. As the character moves through the next levels, it will encounter:

 - This is a tomato. By colliding with the bottom of it, the player lives increase to five and the tomato darkens.

At one of the levels, the player is required to use the mouse. In order to activate the feature, the player must walk to the right until reaching the tile marked with a 1, like the following image shows:

As the player steps on the tile, the instructions appear on screen and the mouse makes itself visible. The player must left-click anywhere inside the tile above the “1” and drag the mouse in a straight (or nearly straight) to the right, never releasing the left mouse button, until reaching the tile above the “2”, which creates a bridge. By releasing the left button too early or dragging the mouse in an incorrect movement, the mouse will place itself at the beginning point.

By doing the correct movement, the following image appears:



At any point during the game, by pressing the “Escape” key in the keyboard, the following menu appears:

continue - The game continues, as if it was never stopped

restart - Restarts the game

quit - Quits the game

In the last level, the player will encounter the following tile:

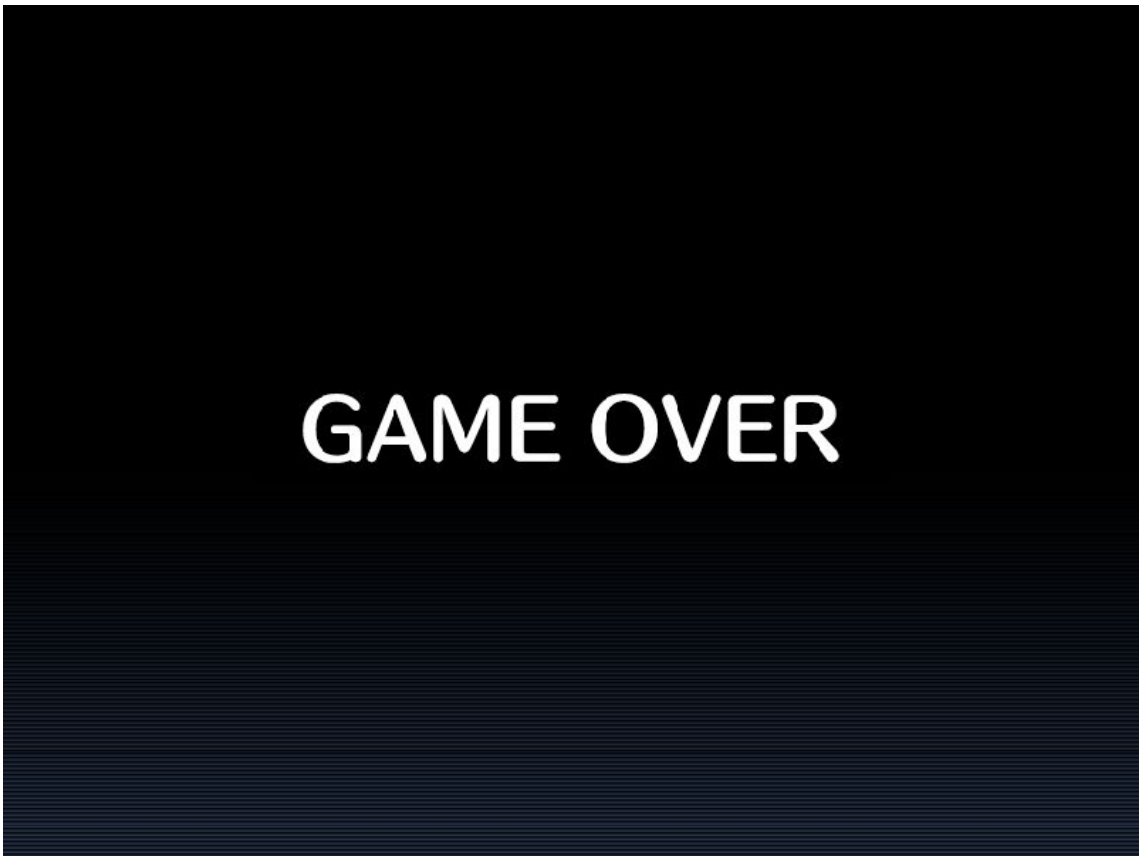
This is an end flag. By colliding with it, the playthrough ends and the final menu appears.

When the playthrough is finished, the following menu appears:

restart - Restarts the game

quit - Quits the game

By quitting the game, at any point the following image appears:



Afterwards, the program ends.

Project Status

Device Table

Device	Usage	Interrupts
Timer	Controlling the frame rate and when to read time from the RTC	Y
Keyboard	Controlling the character movements and exiting to pause menu	Y
Mouse	Used to select options in the start, pause and final menus and to draw an in-game bridge.	Y
Video Card	Displays all on-screen graphics display	N
RTC	Get real time to show on menus	N

Timer

Used to generate interrupts at a set frequency, which is used to update the screen and game logic.

gamePlay::ih() – Handles the timer interrupts and sets the frequency in which the game will be played, by using functions defined in timer.c

devices::timer() – Updates the game

Keyboard

Used for player movement and to exit to the pause menu.

gamePlay::ih() – Handles keyboard interrupts, by using functions defined in keyboard.c

devices::keyboard() – Sorts the scancodes and decides what the game must do

Mouse

Uses both position and buttons.

Used to select options in the several menus and to draw a bridge in a particular level.

gamePlay::ih() – Handles the mouse interrupts and the enabling and disabling of data reporting, by using functions defined in mouse.c

devices::mouse() – Sorts the keyboard inputs by checking what must happen in which situation, by using functions defined in mouse.c

Video Card

Used to draw pixels on screen. Uses 0x115 video mode, a 24-bit direct colour mode, with a resolution of 800x600. Used to get mode information and initialize video mem where image colour information is written to.

A modified version of triple buffering is used where there are multiple 'third' buffers with the background, start, pause and end menu screens. Depending on the game state one of these is copied to the main buffer where other graphics (e.g. entities, cursor) are then drawn. This is done to cut down on proccessing time from loading and drawing large xpm maps to make the game run smoother at the cost of more memory used.

There is collision detection between moving entities. Animated sprites are present.

Using VBE functions 0x01 and 0x02.

gamePlay::ih() – Changes to graphic mode and back, using functions defined in graphics.c

devices::timer() – Draws the current state of the game, by using functions defined in graphics.c, game.c and entities.c. It also checks for collisions, by using functions defined in game_physics.c

RTC

Used to get the real time.

gamePlay::ih() – Updates the real time of the game, by using functions defined in rtc.c

Code Organization/Structure

Tiles and maps

Create tiles and tile map that will then be used to draw the background and for collisions.

Tile maps are created from a tile_char_map_t (char arrays that will be used for to create tile_map_t's)

Typedefs used to set the size of the 2D char array:

- tile_char_row_t
- tile_char_map_t

tile_type - enum used to set different types of tiles (used for graphics and collisions)

tile_t, *tile_p - struct with tile information (x and y coordinates on the screen, hitbox width and height and tile

tile_map_t, *tile_map_p - struct with an array of tiles that represents the entire tile map

Developed by: André Flores

RTC

Basic module for use of the RTC according to the functionality needed in the game.

Developed by: André Flores

Project Utils

Handy utility functions specific to the project

Developed by: André Flores

Timer

Similar to lab2 version.

Developed by: André Flores

Mouse

Similar to lab4 version. Added functions to handle the drawing of the bridge.

Developed by: Diogo Faria

Keyboard

Similar to lab3 version.

Developed by: Diogo Faria

Graphics

Basic functions to change graphics mode, get graphics information and draw xpms.

Developed by: André Flores

Gameplay

Handles the overall game interrupts and within it, the various devices used in-game.

Developed by: Diogo Faria

Game

Game struct and functions that alter game.

*Mouse_p – struct that contains mouse information

game_state – enum with the various states the game can be in

game – struct with all the game information

Developed by: André Flores(game struct), Diogo Faria(Mouse_p struct, game struct, game_state enum)

Physics

All physics used in the game, including collisions.

Collision_type - enum listing the various collision types (collisions will be handled differently for each)

Developed by: André Flores

Files

Used to read and write top score file

Developed by: André Flores

Entities

Contains all entity information and functions to update it.

entity_type - enum with the different entity types

entity_movement_state - enum with the entity´s state of movement

entity_direction - enum with the entity direction

entity_animation_frame - enum with the current entity animation frame

Entity, *Entity_p - struct with all needed entity information

Developed by: Diogo Faria(update_entity_anim() function), André Flores(everything else)

Devices

Functions

Developed by: Diogo Faria

AnimSprites

Inicial files in the LCOM repository, then altered.

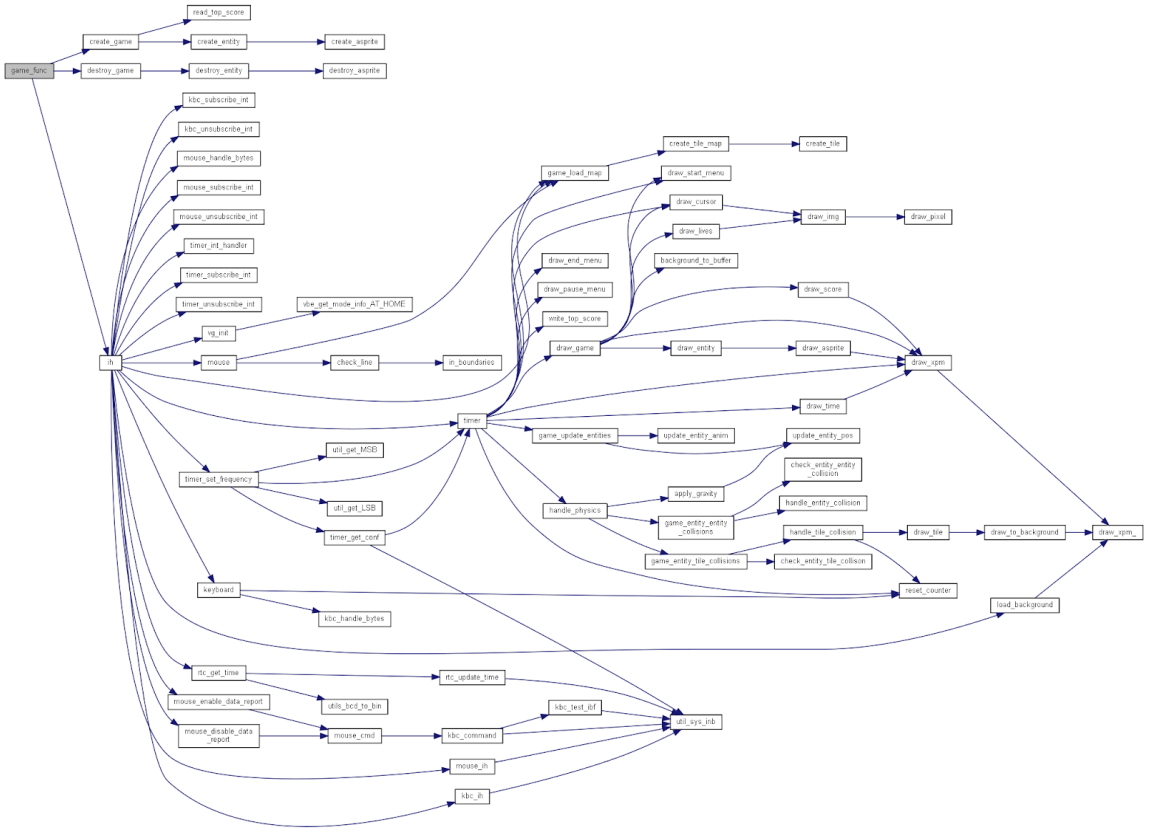
Basic animated sprite struct and functions.

Altered by: André Flores

Relative weight

Tiles and maps	6%
RTC	1%
Project Utils	1%
Timer	5%
Mouse	6%
Keyboard	6%
Graphics	9%
Gameplay	3%
Game	20%
Physics	15%
Files	2%
Entities	10%
Devices	15%
AnimSprites	1%

Function call graph



Implementation Details

All entities in the game are state machines. An entity's state is calculated according to its speed vector and number of lives. User generated events only alter the playable character's state indirectly then, since physics (e.g gravity, collisions) are also present in the game.

An entity can collide with another entity or with a tile from the tile map of the game level it is in. Both types of collisions are checked using hitboxes. The creation of the tile maps used in the game levels from char arrays allows for easy editing of the maps for debugging and game level creation. This also makes it so there is no need to edit level specific background images since each level's background is created at runtime by overlaying each tile's graphic over a generic background (used for when a tile is transparent).

To make the game more efficient we make use of the 2D array in tile_map_t and an entity's position to determine in which tile the entity is. The largest entity can be in contact with at most 6 tiles at a time, so only 6 tiles are checked for each entity each time. This way we don't need to check collisions with every tile present in the game at a time (there are 192). When an entity collides with another entity, no more collisions are checked for that entity in that call of game_entity_entity_collisons() for the same reason.

Also to make the game more efficient, the minute value is only checked once every 60 seconds and the hour value once every 60 minutes from the RTC (these require sys_outb and sys_inb use). This is done by using the number of timer calls since the RTC is only used to read the current time.

Our own version of vbe_get_mode_info() was implemented and is present in graphics.c.

The game structure itself is also a state machine. Its state is changed by the events of the game, like keyboard inputs, character deaths and collisions, and will decide what is going to be shown on screen or how in-game variables, like entities, will be updated. Furthermore, the game structure contains all the pertinent information, like the score, the information regarding the input bytes for the keyboard and mouse and others.

A structure was used as it makes it more efficient to create different entities of a game and to change its settings more easily (speed, number of entities, etc), as well as to be able to represent the game in a more comprehensible form.

Course Evaluation

It will be included in the self-evaluation form.