The game:

We have implemented a variant of “Crayon Physics” with Box2D and OpenGl. We have extended the game with various usability features and some additional graphics details. In this file we describe how to run the game.

Running the game:

You can run the game by cd-ing into the folder containing *main.cpp* and running the game as *./main* . The window presents you with the first level (five levels in total) and a ball. The finish is represented by a blue circle (which is obviously not affected by physics).
The game does not start automatically. To run the game, press the ‘s’ button. If you succeed, you will go on the the next level and your score will increase (see toolbar). If you do not succeed, the level will restart and some value is subtracted from your score. If, for some reason, the ball gets stuck you can restart the level manually by pressing ‘r’ (again, some value will be deducted from your score).
Some levels require you to add objects to the level. To do this you can click four points on the screen. After having selected the four points the game will spawn a new object. Note that this object is affected by physics and will therefore, most likely, not stay where you put it. However, adding objects to the level will have a penalty: If you add large objects more points will be deducted from the score compared to when a small object is added (more on the scoring system can be found under *extra features*).

Commands:

- **q:** Quit the game (provided feature).
- **s:** Start the game (cannot pause game while it is running).
- **r:** Restart the level.

Extra features:

We have added two extra features to the game.

The first added feature is the an animating background color. The background animates between random colors every 100 frames. It does this by interpolating between the base color and the next color. The code for the feature is placed inside update_state().

The second added is a scoring system. The scoring system works as follows:
- The player starts every level with a fixed level score based on the difficulty of the level.

- The player loses 100 points of their level score on death.
- The player loses points of their level score when making a new object based on the size of it and the amount of objects already made.
- The level score can not go below 0.
- The level score is added to the total score when the level is completed.

A player can see their current score and level score in the title bar.

Code outline:

Because the code has been written by us (and not guided by code stubs), we outlined in the code what a block of functions does. To make it more clear we describe the functions that we use here (at a high level) to get a better understanding of what is going on:

In the first part of the code we set up a bunch of variables. These are of three types:

1. Physics parameters required for Box2D to function, such as gravity, size (e.g. ball_radius), friction, densities, number of velocity iterations, etc.
2. Parameters required for drawing, such as colors and references to vertex buffer objects.
3. Game parameters and current game state, such as current level id, window size, variable indicating whether the game is paused, number of selected vertices by the user, references to objects currently present in level, and variables for the scoring system (i.e. rewards & penalties).

Once the setup has been done we define a set of functions for initializing the level, drawing and logic/mainloop:

init_ball():
Initializes the ball as a box2D object (with a body, CircleShape and fixture) and adds it to the world. It also sets up the OpenGL VBO for the object as a continuous array of vertices. This function is called everytime we start up the game or we advance to a new level.

init_objects():
Initializes the objects provided in a level_t struct (depends obviously on what level you play). It clear the previous level (if it exists) and creates new Box2D bodies (keeping a reference). The objects are dynamic or static depending on their type. If needed, the objects may also be joined by using Box2D's joints.
Additionally, the function sets up the OpenGL VBO for the objects present in the level. This function is called everytime we start up the game or we advance to a new level.

init_finish():
Initializes a OpenGL VBO for a circle (representing the finish) identical to how we initialize the ball (only differing in the amount of vertices used and its radius). Note that no Box2D

objects are created, because the finish is not represented as a physics object. This function is called everytime we start up the game or we advance to a new level.

load_world(int id):
Load the world that is connected to the given id. It calls init_ball, init_objects and init_finsih to set up the level. It also sets up the current level score multipliers (see section on extra features) and pauses the game. This function is called everytime we start up the game or we advance to a new level.

sort_indexes(vector<float>):
Returns a vector<int> with indexes of the given vector<float> sorted by the value of the floats from small to big.

add_new_object():
Adds a new quadrilateral object based on four coordinate points that are provided by the user.

draw_ball():
Draws the ball in the frame buffer. It asks box2D for the balls position in the world and draws it according to the contents of the VBO as a triangle fan (but translated to position it correctly on screen).

draw_objects():
Loops through the objects in the level (together with the objects created manually by the player), asks for the current position and orientation of the objects from box2D and draws the contents of their respective VBOs translated and rotated according to what was provided by box2D.

draw_finish():
Similar to draw_ball(). If looks up the location of the finish and draws the circle stored in its VBO at the correct position.

update_state():
this function manages the logic of the game and is called every frame. It does the following tasks:

1. It checks whether the ball is still in the frame. If it is not (e.g. fell through the ground), then it will restart the current level and remove points from your score.
2. It checks whether the ball has reached/touched the finish. If there are more levels to go it will simply transport you to the next level. If it was the final level, then it will display your final score in the toolbar.
3. Animates the background color and draws the finish, ball and objects in the scene.

resize_window():
Gives opengl new window coordinates (provided by framework).

key_pressed():

Handles the different possible key presses that can be used for playing the game (the start key, restart key and the quit key).

mouse_clicked():
Handles the mouse clicks so it can be used to create quadrilaterals / objects in the level. It registers the points that are clicked and converts them to the world-coordinates of box2D. If four points have been selected it calls add_new_object to initialize the object in the level.

mouse_moved():
Does not do anything in our implementation, but is kept because it was provided by the framework.

main():
Sets up the GLUT window, keybindings and OpenGL state machine w.r.t. the vertex operations (modelview matrix). It is called when the program is started, and initializes the game loop.