2D Physics Puzzler Report

Game Summary:

The basic objective of this game is to ensure that a ball (dynamic circle), safely reaches a goal zone on the other side of the screen. If the ball is simply released with no other action, the ball will fall and be trapped in a ditch before the goal, forcing a restart. To prevent this the player must drop a block swinging from a pendulum into the ditch, making a bridge for the player to cross.

Design Decisions:

The program implements the Box2D physics engine in order to calculate collision between different entities and how they behave as a result. With Box2D's built in physics, it was much easier to construct more complex interactions between objects, using things like joints, sensors etc. Also included is the Simple Fast Multimedia Library (SFML), to provide sf::Drawable as an inheritable class and give objects a visible form.

Not much is included in the main.cpp file, though we can see the window being drawn, and the creation of a game loop, which contains a call for an update function from within the "Game" class, which associates with all other classes.

Game.h contains class objects of all the other classes, which allow access to all functions from those classes. This class also handles input from the player, from either a mouse click or button press, as well as creating two sf::View objects for game objects and UI text. There is also a b2World object, and functions for updating and drawing objects and text.

There are two types of block classes, the "StaticBlock" class where two blocks are created to serve as anchor points, and the "DynamicBlock" class where the pendulums, the hole-filling block, and the b2Joints that hold the pendulums to the static blocks. Since the dynamic blocks need moving, they of course need an update class, there are also functions for deleting joints and resetting block positions, which are called in the game class. When resetting the block's positions, it is important to delete the original bodies, to prevent any invisible duplicates from staying in the game world.

The same applies to the "DynamicCircle" class, which is similar to the dynamic block class with an update and reset functions, but it also contains a function that returns the body like the static block class, so its position can be used as an anchor point for a b2Joint.

The “SensorContactListener” and the “HitSensor” class together allow the game to check that the ball has reached the goal. The HitSensor class creates a sensor for the SensorContactListener to listen to and, while not necessary, it gives the sensor a visible form, making it easier to check that the ball is visibly touching it during testing. HitSensor also contain a couple of function called by the listener and Game classes, that return true when the ball reaches the goal. When this happens, a congratulatory message is displayed.

Lastly we have the “Terrain” class, which forms the ground the ball rolls along. The terrain is made up of 28 points in a vertex array using sf::TriangleStrip, as well as 2 for loops which colour the ground and add collision. The formation terrain is made with 3 distinct parts, the slope at the beginning for the ball to gain momentum, the pitfall near the middle which the player will need to cover with a block to prevent the ball from getting trapped, and the goal zone, marked by the visible sensor.

Testing

Method:

The game followed a very test-driven development, these tests range from simply drawing an object to the screen, to testing the interactions between certain objects. The tests have been organised in the chronological order they occurred, shown by their id. The expected outcome was written with the actual results in the neighbouring column. If there were any fixes found for an irregular outcome, or any additional adjustments to the code following the test, they have been documented in the comments column.

Any simple errors like spelling mistake, or very basic errors in the code that caused a repeat of the test have not been documented, or addressed in the comments column, to keep test documentation concise and easy to read through. Any tests which have the same expected outcome as the previous test have been marked with '~'.

ID	Description	Expected outcome	Actual outcome	Comments
1	Gravity	Draw dynamic circle to the screen and have it obey Box2D gravity.	Circle was successfully drawn and fell from top to bottom according to gravity values.	No comment
2	Draw terrain	Draw terrain with vertex coordinates and fill with colour.	Terrain was somewhat scrambled, with little resemblance to the given coordinates.	This was caused by some points in the vertex array not having assigned values (fixed).
3	Terrain collision	Terrain will be given collision, allow the dynamic circle to bounce off it.	Collision occurred but not as expected, ball bounced after passing halfway through the terrain.	The for loop drawing the terrain has been altered, this seems to have fixed the issue.
4	Dynamic block draw	Draw dynamic block to the screen and have it obey box2D gravity.	Block was drawn and falls correctly but collides inaccurately with other objects.	Weird collision caused by the origin being off centre (fixed). Another dynamic block was drawn in the same way.
5	Joints	Connect 3 objects with Box2D joints, a static block, and 2 dynamic blocks.	All blocks are connected, with the static block as the main point of rotation, the middle block acting as a pendulum, and the other dynamic block swinging from the other end of the middle one.	Block position coordinates were changed numerous times to make connect the blocks in a way that they would swing more realistically.

6	Hit sensor	Victory text will be displayed when the ball hits the sensor.	Nothing happened when the ball collided with the sensor.	Hit sensor required a contact listener, which had no class.
7	Text display	Text will be displayed when initially set to a value other than transparent.	Text was not shown on screen.	Text required a separate text view in order to be displayed correctly.
8	Hit sensor 2	Victory text will be displayed when the ball hits the sensor.	Victory text was displayed upon collision.	Hit sensor was moved to a more convenient position for the purpose of these tests.
9	Joint destruction	A joint will be destroyed when the blue pendulum is clicked, dropping the block attached to it.	The joint was successfully destroyed, causing the block to fall.	A similar joint set up was made for the dynamic circle, it works the same way.
10	Game completion	Guide the ball into the goal to see if the level is completable.	Level is completable but is somewhat unintuitive with how the dynamic block is dropped into the pitfall.	Altered the dimensions for the dynamic block, to make it easier to drop into the hole.
11	Reset	Upon hitting ‘R’ on the keyboard, game objects will reset to how they appeared at the start.	Objects visibly reset for a frame, before returning to their positions before the key was pressed.	The body positions may have to be changed along with the shapes.
12	Reset 2	~	Objects reset but seem to collide with invisible objects.	Old bodies must be destroyed and remade when hitting the reset key (fixed).
13	Reset simplify	Find out which lines of code currently in the reset functions are unnecessary for the reset to work.	All lines involving the SFML objects aside from “setPosition”, are unnecessary. All lines involving the bodies must be included.	As the bodies are destroyed out of necessity, all properties of the bodies must be reassigned.
14	Break joint after it’s broken	Break a joint after it is already broken to see what happens.	Game crash, clicking it again causes the code to check for a joint that no longer exists.	Changed the contents of the joint when clicked instead of destroying.