Author: Tony Kingston

Controls:

Arrow keys to move when locked onto the player sphere. Shift/Space can be used to move up and down if desired.

WASD to move camera, Shift/Space to control height

G to control gravity on/off. P to control sleeping on/off, B to control broad-phase collision detection.

Keys 1, 2 3 and 4 will load various test scenarios.

Press N to toggle showing bounding boxes (currently only implemented for AABB and OBB)

A video should be available here: https://github.com/TonyKingston/85023Video

Here is a list of some of the things that my solution contains:

Menu

* Game opens with a menu allowing the player to select between the physics level or the maze level.
* A pause menu can be accessed in game by pressing F10
* A pushdown state machine is used to manage the state of the menu and the game.

AI

* Maze contains an AI controlled enemy that uses a behaviour tree.
* The AI can generate a path, using grid-based pathfinding, and follow it.
* The AI can go after the player or collect the bonuses (currently the AI has no knowledge of if a bonus has been collected by the player instead, so it will still travel to its location)
* The AI respawns if it touches the player.
* After following its path to a bonus, the AI will send out a ray. If it finds a bonus it will travel towards it. This helps avoid the AI missing the bonus.

Collision

* AABB/AABB, AABB/Sphere
* Sphere/Sphere
* OBB/Sphere, OBB/OBB
* Ray/Sphere, Ray/AABB/, Ray/OBB, Ray/Plane, Ray/Capsule
* Clicking will cast a ray, showing the debug information of any object it collides with
* Objects can also fire out rays when clicked on
* A quad tree is used for broad-phase collision detection. Nodes are put to sleep if there are no active objects in them. The Raycast method in GameWorld uses the quad tree to get likely collisions against the ray. I meant for static objects to be processed separately, but it’s only partially implement and needs more debugging.
* Objects have their own layer masks (currently Default, IgnoreRaycast, Player) which determine if a collision is possible with another object e.g. only players and enemies can collide with bonuses.

Collision Resolution

* Projection and Impulse used to resolve collisions.
* Friction is applied to objects
* Objects have their own restitution and friction coefficients. Combined by taking the average.
* Penalty method implemented, a sphere object in the physics course uses it.
* Trigger volumes used to change the game’s state upon contact with the player
* The average motion of each object over time is calculated. If not above a certain threshold, the object is put to sleep and collision/resolution no longer occurs. (P to switch on/off)

Constraints

* Position constraint used to constrain pendulum obstacle.
* Pendulum also uses an orientation constraint to lock its rotation in a desired axis.

Player

* The player sphere can be controlled using the arrow keys.
* Numbers 7-0 can be used to add torque to the player if desired.

Gameplay

* Bonus objects can be collected for points
* Player loses if lives run out. If playing the maze, the game will last for 90 seconds.
* Player wins if the get to the end of the physics level
* A results screen shows the player’s score and time taken and enemy’s score if applicable.