CAB202 Tutorial 3
Collision Detection

For assignment one and many non-assessed questions in CAB202, we deal with some form of movement in a character or sprite. In order to simulate realistic movement and collisions, we require an algorithm to handle collisions between objects in an ncurses terminal. This tutorial will help you develop this functionality through the practice non-assessed question.

ZDK Sprites

In the AMS questions for topic two, we investigated the use of the ZDK Sprites library in order to create a "zombie" avatar for our player. The sprites library provides a powerful set of functions and structures that make developing a collision detection system much simpler.

```
double sprite_x(void) : returns the x coordinate of the sprite

double sprite_y(void) : returns the y coordinate of the sprite

double sprite_width(void) : returns the width of the sprite

double sprite_height(void) : returns the height of the sprite
```

In particular, the ability to return and store a sprites (x,y) location on the game screen, and its width and height – as shown above (you can read more information about all the sprite functions in `cab202_sprites.h`) – give you all the information you need to implement a simple collision detection algorithm for our ncurses game.

Non-Assessable Exercise

NOTE: This exercise is not assessable, and the tutor can help you with it. Assessable exercises are in the Automatic Marking System.

1. Wrecking Ball Game

The tutorial file **question_1.c** on blackboard contains a modified version of the Zombie Dash Jr. game developed during the lectures. It has been modified so that there are 3 sprites; a 2x2 hero sprite (comprised of the 'H' character), a 2x2 "wrecker" sprite (comprised of 'x' characters), and a 1x1 "goal" sprite (a single 'o') character. Your job is to implement the three functions declared in the question_1.c file, in order to build the game functionalities.

Functions to implement:

`void collision_check(void)`: Using the ZDK functions for sprites, you will need to determine if any of the three following conditions are met:

- Player has collided with the goal: If true, increment the variable hero_score by 1, and call the reset_goal function
- Wrecker has collided with the goal: If true, increment the variable wrecker_score by 1, and call the reset_goal function

- Wrecker has collided with the player: If true, increment the variable wrecker_score by 1, decrement the player_score variable by 1, and call the reset_player function

`void reset_hero(void)`: When this function is called, move the hero sprite to a random location within the game play area (HINT: Look at the setup_wrecker function)

`void reset_goal(void)`: When this function is called, move the goal sprite to a random location within the game play area (HINT: See above)

Challenge Exercises:

- *Modify the program so that the wrecker deflects off of the player sprite after a collision, instead of resetting the hero's location*
- *Add additional wrecker or goal sprites. Is there a more efficient way to store multiple sprites? How could one use loops to make the code smaller and more effective?*
- *Investigate the use of mathematical models of collisions (found in physics subjects) to create realistic deflection between wrecker sprites. (HINT: Find code/help here - http://gamedevelopment.tutsplus.com/tutorials/when-worlds-collide-simulating-circle-circle-collisions--gamedev-769)*