2CAB202 Assignment 1: Zombie Jump

Due Date: 11:59PM Sunday 24th April 2016
Submission: via AMS
Marks: 30 (30% of your final mark)

For this assignment, you will be writing your own version of a game called Zombie Jump. This game is inspired by the popular mobile platform game called Doodle Jump (Figure **1**). The aim of the game is to move a player between platforms.

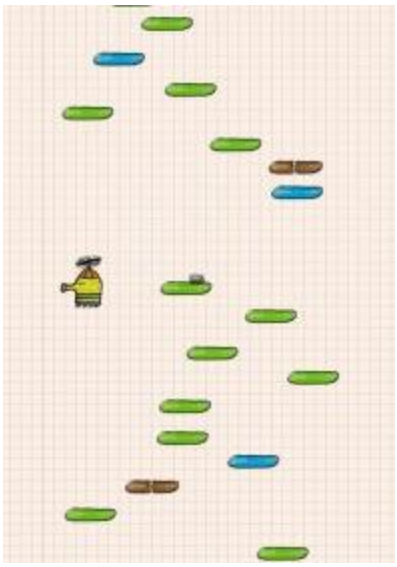

Figure 1**. Example screenshot of game Doodle Jump**

Requirements

Your task is to implement the Zombie Jump game. Throughout the semester, you have been provided with a number of examples of skeleton code for implementing games with the CAB202 ZDK library. You have also been shown a demo version of the game (*see the week 4 lecture*), which implements some of the requirements for the assignment. You are not required to use the ZDK library, but you are strongly encouraged to do so.

WARNING: QUT takes plagiarism very seriously. If your assignment has material which has been copied from other class members, previous assignments, or code from any other source you will be penalised severely. If you cannot explain exactly what your code is doing, it is clearly not your own work, and you should not be submitting it as such!

Game Specification

The aim of the game is to accumulate as many points as possible, while staying alive for as long as possible. Blocks (platforms) appear on the screen moving upward (scrolling up). Some blocks on the screen are safe and some are forbidden. The player should always jump between safe blocks gaining one point every time he/she lands on a block. If the player touches a forbidden block, a life will be lost and the player

restarts on the 'top block' at the bottom of the screen. The blocks scroll up towards the top of the screen at a uniform speed (see below).

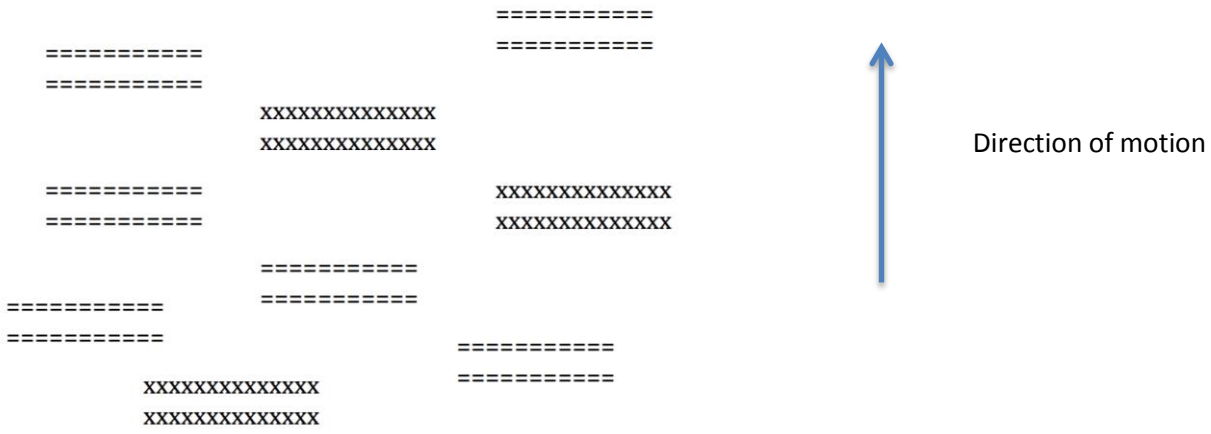

Figure 2. Example block configuration

For all levels, the player can move horizontally while on the block. The motion of the player in level 1 is downward with a constant velocity (i.e. free fall), and both upwards and downwards under the influence of gravity in levels 2 & 3. Level 3 adds controllable block speeds, and animated sprites that attempt to kill the player.

A score and level (bottom left), timer (top right), and remaining lives (top left) should be displayed on the screen at all times. When all lives are lost, the game ends and a Game Over screen should be shown, giving the player the option to restart or quit. If you have completed more than level 1, **you must be able to toggle between levels in game with the 'l' ('l' for "Level") key to receive any marks for the extra levels.**

Configuring your screen size for the game

This game is suited to a game screen with a portrait orientation (rather than the default landscape terminal of 80x24). For this assignment, your game must be able to run at a resolution of 55 character rows by 70 character columns. To change your terminal resolution from the terminal, type one of the following commands:

echo -en "\e[8;55;70t"

resize –s 55 70
(use this if the first command fails – i.e. in OSX)

Level 1: Basic functionality [14 marks total]

All versions require the following basic functionality (unless it is superseded by a feature in a later level). Unless this basic functionality is complete, you will not be eligible for any marks in the later levels. Marks are broken down as follows:

1. The game display screen is entirely visible at all times (i.e. never occluded in any way) and display: **[1 mark]**
    a. Lives remaining (top left)
    b. Playing time in minutes and seconds format (top right)

    c. Level and score (bottom left)
    d. Boundaries around the top and bottom displays that clearly define the bounds of the game area
2. The player initially appears on a safe block at the bottom of the screen (with no other blocks on the screen). The player's sprite is 3 characters high and one character wide **[1 mark]**
3. All blocks are **[2 marks]**:
    a. 2 characters high
    b. At least 7 characters wide (they can all be the same width for level 1)
    c. Always at least 4 characters horizontally, and 3 characters vertically, away from other blocks
4. Blocks randomly appear from the bottom of the screen. They: **[3 marks]**
    a. Don't appear in explicit rows
    b. Have no consistent, observable pattern
    c. Each appear in a randomly selected column
5. Blocks move up at a speed of 2 characters per second, and disappear when they hit the top of the game area **[1 mark]**
6. There are at least: **[1 mark]**
    a. 5 safe blocks (denoted by '=') on the screen at all times (when screen is full)
    b. 2 forbidden blocks (denoted by 'x') on the screen at all times (when screen is full)
7. Player movement behaves such that: **[3 marks]**
    a. Left and right arrow key presses move the character left and right respectively.
    b. It is impossible to move outside the edges of the screen.
    c. When on a block, the player moves up at the same speed as the block.
    d. When off a block, the player free falls downwards at 2 characters per second.
    e. No lateral motion occurs during downward motion.
8. The following game mechanics are present: **[2 marks]**
    a. The player dies when they fall off the bottom of the screen, hit the top of the screen, or hit a forbidden block. On death, the player starts from the bottom again with only one block on the screen.
    b. A point is scored every time the player lands on a safe block.
    c. When there are no lives remaining, the game can be restarted by pressing 'r' (score, lives, time, and screen all reset) or quit with the 'q' key.

Level 2: Parabolic motion [8 marks total]

Level 2 follows the same specifications as level 1. The behaviour specified for level 2 overrides any conflicting behaviour from level 1. The block widths are no longer fixed, and player motion now allows jumping and falling in parabolic paths (the downwards velocity from the previous levels is replaced by gravity). Marks are broken down as follows:

1. Each block has a random width between 3 and 10 characters **[1 mark]**
2. Player motion is controlled as follows: **[3 marks]**
    a. A left or right arrow press gives the player a horizontal velocity.
    b. When on a block, an up arrow key press gives the player a vertical velocity upwards.
    c. When on a block, a down arrow key press sets the player's horizontal velocity to zero.
3. A constant vertical acceleration (i.e. gravity) acts downwards at all times, generating a reasonably realistic ballistic motion effect when the player is in flight after stepping or jumping from a platform. Gravity generates the following **visually observable** behaviour: **[3 marks]**

a. When on a block, the effects of gravity are eliminated – the player does not fall through the platform.
b. When jumping, the player's vertical velocity is slowest at the peak jump height.
c. When jumping with a horizontal velocity, the player traces a parabolic path. The horizontal velocity is constant, while the vertical velocity increases at a constant rate with passage of time.
d. When falling with a non-zero horizontal velocity, the player also traces a parabolic path but starts with zero vertical velocity.

4. When hitting the side of a block, the player's horizontal velocity becomes zero **[1 mark]**

Level 3: Advanced Features [8 marks total]

Level 3 follows the same specification as the previous levels. The behaviour specified for level 3 should override any conflicting behaviour from previous levels. This level adds 3 different speeds with smooth transitions, and animated "destruction sprites" of varying sizes. Marks are broken down as follows:

1. The vertical speed of the blocks has 3 different levels: slow (0.25 times level 1 speed), normal (level 1 speed), fast (4 times level 1 speed) which is changed by pressing the number keys '1', '2', and '3' respectively. This effect applies only to blocks; the behaviour of the player remains unchanged, moving as normal at all times. **[0.5 marks]**
2. When changing between two different speeds, there is a smooth transition between the current speed and requested speed. This smooth transition is **visually observable**, and while happening all other speed requests are ignored **[2 marks]**
3. The game is completely functional at all speeds (and while transitioning), and displays the speed in the bottom right corner. It displays "SLOW", "NORM", "FAST", or the current transition (e.g. "SLOW -> FAST") **[0.5 marks]**
4. After a random delay, a circular sprite enters the screen. It travels in front of the blocks, and kills the player if they fall under any part of the circular sprite. The sprite travels in a random diagonal line across the screen. While travelling the line, the sprite performs one circular loop on screen, and continue on the line **[2 marks]**
5. The sprite is animated to reflect the direction it is currently travelling. Each character in the circle is a '>' when the direction is closest to left, '^' when closest to up, '<' when closest to right, and 'V' when closest to down **[1 mark]**
6. Each time the sprite comes on screen, it is a random circular size with a minimum radius of 3 and maximum radius of 15. The bitmap for this sprite is **dynamically allocated**. The submission form points to code that allocates the memory, and demonstrates that the memory is freed without memory leaks **[2 marks]**

Marking

The assignment will be out of 30 marks and will be worth 30% of your total grade in this subject. The breakdown of marks is outlined in the task specification above. The following should be noted about marking:

- **If your code does not compile, you will get 0 marks for the entire assignment.**
- If segmentation faults occur, you will receive marks for what was displayed but lose marks for the segmentation fault. No more of your assignment will be marked. We will not debug your code to make it compile or run.

- Your game must be easily playable. If timings, settings, or controls are set in a manner that makes it difficult to play (e.g. not using the key inputs specified, ridiculously fast movement, etc.), you **will receive 0 for the assignment**.
- Mark penalties will be applied if the code exhibits general defects or undesirable behaviour. This includes, but is not limited to, errors in object motion, such as objects jumping more than one unit per frame, or sprites overlapping when they are not required to by specification.
- Your marker will not spend longer marking your assignment if they cannot easily demonstrate a specific feature. It is **your responsibility** to demonstrate that a feature works, not the marker's to prove it is there.

Submission

Your assignment is due on Sunday April 24th 2016 at 11:59pm. Submission will be online through the AMS used in the tutorials. You must submit through the submission page (available at http://bio.mquter.qut.edu.au/CAB202/Exercise?TopicNumber=6&ProblemNumber=1), and follow all of the instructions. Submission will consist of the following documents (all of which must be submitted as part of your AMS submission):

- Your complete source code.
- A ZDK movie transcript which demonstrates your program to the marker. The movie must play correctly in the CAB202 Movie player tool (http://bio.mquter.qut.edu.au/CAB202_movie_player/). Follow the instructions on Blackboard under Software Resources to create a movie. The **maximum permissible duration of your movie is 5 minutes** (anything after this time will not be marked).
- A filled-in copy of the Statement of Completeness spreadsheet:
    - A template will be provided for you to fill out and submit.
    - You will use the Statement of Completeness to document all features that you have implemented. Documentation of a feature consists of the following:
        - Explicit declaration of the files and line numbers that correspond to your implementation of the feature.
        - An accurate indication of the times (measured in seconds from the start of the movie) at which the feature is visibly implemented in your movie.
    - Marks will only be awarded for the features which are marked as complete, and adequately documented, in the Statement of Completeness. No marks will be awarded for undocumented features, or features with vague or incorrect references to source code locations or movie demonstration points.

When submitting to the AMS, it is your responsibility to make sure that your assignment compiled correctly. The AMS will compile your code and return any errors that occur. As mentioned above, if you do not submit a compiled assignment, you will receive 0 marks. If any part of the documentary evidence described above is missing, you will receive 0 marks. You will have unlimited submissions of the assignment, so there are no excuses for not resolving any compilation issues.