**Game Specification Form Student ID: wgqp43 Level 3**

|  |  |
| --- | --- |
| **Marking Criteria** | **Describe how your game matches the criteria** |
| **Game design (10%)** | |
| Game Goals: | Game goals:   * Collect all 10 treasures * Try and get the highest score possible * Avoid the enemies |
| Game Type: | 2D Platform Game |
| **Core development (30%)** | |
| Game scene (visual representation [2D, 2.5D or 3D], internal data structure): | A 2D platformer with randomly generated infinite level that gets progressively more difficult the further a user |
| Game flow / game progression (e.g., navigation, screen scrolling, levels): | Game flow: Once the user reaches a certain point on the screen the screen scrolls downwards revealing more of the level above. The level is infinite so this will continue until the player ‘dies’.  Game progression: As the game progresses, the level gets more difficult, with more enemies, fewer powerups, etc. |
| Game interaction (e.g., action detection and response generation): | User input: Players can move a character with keyboard inputs, allowing him to jump, crouch, move, etc.  Level generation: Once the player reaches a certain part of the screen, the screen will scroll and further level information will be displayed, with previous level information being removed (that is out of bounds). |
| Game object (e.g., use of sprite, 3D objects, animation, multimedia): (DONE) | Sprites: Sprite classes were used to contain: Players, Platforms, Enemies, Projectiles, Flags, Power-ups  Animation: Most sprite classes had some form of animation, with the player being the most complex animated entity – including animations for idle, walking, jumping and crouching. Enemies, flags and flags are have much more visible animation, with power-ups having a more subtle switch in states.  Multimedia: All sprite classes appearances come from spritesheets that are imported into the program. Occasionally dynamic image modification is used to fit the spritesheet image to the sprite (e.g. platforms with different lengths) |
| **Game mechanics (30%)** | |
| Game rules / logics: | Rules:   * The player must attempt to make it as high as they can in the level, whilst collecting as many treasures (in this case flags) as possible (maximum 10). * The player can move left to right, jump and crouch, and will wrap around the screen. * Platforms are generated either randomly or via prebuild sections, where each platform can be either stationary or moving. * If a platform moves then it will bounce off the screen, not wrap around it. * Enemies spawn randomly, becoming more frequent as the game progresses. * Enemies will always be moving and will fire projectiles randomly at the player. * Enemy movement is limited in the x-axis and will bounce off the screen when reaching a limit * Powerups spawn randomly, giving the user various abilities. The spring will boost the user to a further point in the level, whereas the mushroom will grant the player some random period of invincibility. * The user only gets one life per game, there are no second chances. * If the user hits an enemy, falls off the bottom of the screen or gets hit by an enemy projectile they ‘lose’ and the game finishes * Each game a score counter and a treasure counter are displayed. |
| Game challenges: | **Challenges**:   * Be able to jump from platform to platform (with increasing difficulty as the game progresses) * To avoid touching the moving enemies * To avoid enemy projectiles from hitting them |
| **Good use of game engine (15%)** | |
| Choice (pyGame, Unity): | Choice: pyGame was used for this project |
| User input (keyboard, mouse, joystick): | User input: Comes entirely from keyboard inputs |
| Game object interaction (e.g., event triggering, collision detection): | **Collision detection:** Collision detection is plentiful in this game, with masked collision detection being used for accuracy.  Masked collision detection differs from box collision detection, as instead of checking whether too sprite images collide it instead determines whether the objects within these sprite images have collide.  Collision occurs between the player and every other sprite, however, there is limited collision detection between other sprites (since it wasn’t required).  **Event detection:** If the user reaches a certain score, the game will increase the difficulty using a number of metrics |
| Incorporate multimedia content: | Multimedia content was incorporated into the game (both graphics and animation) |
| Other features used (e.g., asset, incorporation of external libraries): | Use of math library for projectile calculations and player movement. Use of random library for random spawns. |
| **Demonstrate creativity (15%)** | |
| Game economy (e.g., support to game type, game feedback, game difficulty): | **Game economy**: As the user progresses through the game, the game makes itself more difficult through the random level it generates:   * Platform spawns decrease, platforms widths decrease and distancing between platforms increase * Chances of platforms moving increases, along with the platform speed * Enemy spawns increase, as well as frequency of projectiles they spawn * Powerup spawning doesn’t decrease (necessarily), but because of less platforms power-ups become rarer. |
| Advanced Interaction (e.g., game physics, object tracking, steering behaviour): | **Game physics**: All game physics was implemented from scratch using equations of motion.  Player: More realistic equations of motion were used for all motion. Gravity was a constant factor on the player which decreased vertical acceleration during jumping realistically and allows the player to ‘fall’. Friction was a factor on the player when moving left or right, which caused the player to not instantly come to a stand still – and in turn limited the movement speed of a player  Projectile: The projectile sprite was the most complex physics to implement. Once a projectile was fired at a player it always had to follow the same path, even when the camera was moved up (i.e. more level was generated). This was difficult to do. |
| **Game optimisation and configurability (50%) [For Level 4 Students Only]** | |
| Include optimisation to enhance game performance (e.g., game related functions, game scene and objects, interaction, rendering, media content): |  |
| Make the game flexible to support making changes (e.g., game scene and objects, game flow / progression): |  |

**What is expected for the above**

**Game design (10%)**

**Goals**: Determine a set of goals

**Type**: Determine game type (Platform Jumping game)

**Game Mechanics (30%):**

**Rules/Logic** Rules and procedures that guide the player and the game response to the player's moves or actions, typically covering:

* Navigation control
* Action control
* Game resource control
* Skill and difficulty control
* Game flow, progression, reward, and ending condition
* Competition and Collaboration
* Lead to Fun

**Challenges**