TECHNICAL SPECIFICATION

Course: Game Development

Project: Pixel Jumper

Members: Phạm Văn Hiếu 2170054

Trần Huỳnh Đức 2162398

INDEX

1. Project Information

1.1 Purpose of this Document

Giới thiệu và mô tả kĩ thuật của dự án. Làm rõ nhất có thể về những điều cốt lõi của dự án "Pixel Jumper".

1.2 Scope Of This Document

- Mong muốn có một cái nhìn tổng quát cả về phần gameplay và phần source code của "Pixel Jumper".
- Tổng quát hoá phong cách của trò chơi, hiểu rõ hơn "Pixel Jumper".

1.3 Document Overview

1. Projec Information
2. Game Framework
3. Game Mechanics
4. User Interface
5. Art and Video
6. Sound And Effect

1.4 Idenfitication

- Mục tiêu: Xây dựng một game indie với phần mềm làm game Unity và ngôn ngữ C#.
- Nền tảng hướng tới: Mobile, hệ điều hành Android.
- Thể loại game: 2D, Platformer, Run & Jump.
- Tên game: Pixel Jumper.

1.5 Relationship to Other Plans

- Phát triển gameplay nhiều level mới và phong phú hơn.
- Mở rộng sang nhiều nền tảng: iOS, web, PS4, Xbox.

2. Game Framework

2.1 Platform and OS

Game Pixel Jumper có các thông số kỹ thuật sau:

- Target Platform: Mobile.
- OS: Android.
- Minimum API Level: Android 5.0 Lollippop (API level 21).
- Target API Level: Android 10 (API level 29).
- Target Architectures: ARMv7, ARM64.
- Aspect Ratio: 16:9 (1280x720).
- Allowed Orientations: Landscape Right, Landscape Left.

2.2 External Code

2.3 Control Loop

3. Game Mechanics

3.1 Game Objects Data

3.1.1 Menu Scene

Game Controller

Là game object quản lý việc save và load data của game, cung cấp reference tới data object cho các game object khác và hai phương thức tắt mở audio của cả game. Các đối tượng FileStream và BinaryFormatter được sử dụng để Serialize và Deserialize dữ liệu.

Game Controller này được xây dựng dưới dạng Singleton, được tạo một lần duy nhất tại Menu Scene và sống tới khi game tắt. Dưới đây là phương thức để tạo Singleton:

```
private void BeSingleton()
{
  if (instance != null) Destroy(gameObject);
  else
  {
    instance = this;
    DontDestroyOnLoad(gameObject);
  }
}
```

Audio Controller

Là game object quản lý việc chạy background music và phát SFX (Sound Effect). Việc thay đổi Background music sẽ được trigger khi scene thay đổi, còn việc phát SFX sẽ do các game object khác trigger.

Audio Controller sử dụng API sẵn có là **OnSceneFinishedLoading**() để thay đổi background music theo name của từng scene. Một tham số là time cho background music của menu scene được lưu lại mỗi khi menu scene bị tắt, để sau đó được chạy tiếp tại vị trí time được lưu.

Audio Controller sử dụng tất cả 11 Audio Source, trong đó gồm 4 background music và 7 SFX.

Audio Controller cũng được xây dựng dưới dạng Singleton, được tạo một lần duy nhất tại Menu Scene và sống tới khi game tắt. Phương thức để tạo Singleton tương tự với Game Controller.

Main Menu Controller

Thực hiện việc chuyển đổi qua lại giữa Menu Panel và Level Panel theo sự điều khiển của người chơi. Menu Panel là menu với hai lựa chọn Play và Shop, đây cũng là menu đầu tiên người chơi nhìn thấy khi mở game. Level Panel là menu cho phép chọn màn chơi và skin nhân vật.

Nếu người chơi sở hữu từ hai skin trở lên thì một game object có tên Selector sẽ active ở Level Panel để người chơi có thể thay đổi skin của nhân vật.

Main Menu Controller còn thực hiện nhiệm vụ khởi tạo các game object Player theo skin mà người chơi sở hữu. Dữ liệu về skin đã mua được lấy từ singleton Game Controller.

Selector

Selector là game object con của Level Panel. Selector chỉ được active bởi Main Menu Controller khi người chơi đã unlock thêm ít nhất một skin mới.

Selector chứa 3 button con bên trong, mỗi button có vị trí trên Canvas trùng với vị trí của skin nên khi người chơi nhấn chọn vào skin thì thực chất là đã nhấn chọn button tương ứng.

Menu Background

Chứa 3 game object Quad bên trong, mỗi Quad sẽ scroll về bên trái một khảng cách khác nhau mỗi frame. Kết hợp với Run animation về bên phải của Player, người xem sẽ có cảm giác nhân vật Player đang di chuyển về bên phải mặc dù Main Camera hoàn toàn đứng yên.

```
private void MoveRightToLeft()
{
  float time = Time.deltaTime;
  quad1.material.mainTextureOffset += new Vector2(time * 0.03f, 0);
  quad2.material.mainTextureOffset += new Vector2(time * 0.01f, 0);
  quad3.material.mainTextureOffset += new Vector2(time * 0.006f, 0);
}
```

3.1.2 Shop Scene

Shop Menu Controller

Đây là game object quản lý toàn bộ Shop Scene, thực hiện việc mở và ẩn các Panel thích hợp khi người chơi tương tác với item có thể unlock, đồng thời gọi các phương thức của Game Controller khi cần thay đổi dữ liệu.

Nếu người chơi nhấn chọn một item bất kỳ có giá lớn hơn số tiền hiện có thì NotEnoughMoney Panel sẽ hiện ra, ngược lại thì Confirm Panel sẽ hiện ra.

3.1.3 Level Scene

Gameplay Controller

Đây là game object quản lý gameplay của người chơi trong Level Scene, thực hiện việc tạo Player theo skin mà người chơi đã chọn, tắt mở Pause Panel và cung cấp các phương thức cho các Button của Pause Panel và Result Panel.

Trong Pause Panel, người chơi có thể thay đổi độ lớn của Scene bằng hai Button Zoom In và Zoom Out, ở đây, thuộc tính **Camera.orthographicSize** được thay đổi.

GamePlay Controller cũng cung cấp phương thức FinishLevel để game object Exit có thể gọi tới khi người chơi hoàn thành level.

Level Info

Level Info là một game object được thêm vào từng Level Scene. Làm một nhiệm vụ duy nhất là chứa các dữ liệu cần thiết cho riêng từng level, như số level hiện tại, tọa độ đầu tiên của Player khi màn chơi bắt đầu.

Collectible Controller

Đây là game object quản lý các collectible của người chơi trong từng level, cung cấp các thông tin về số lượng collectible và tổng money hiện tại.

Death Controller

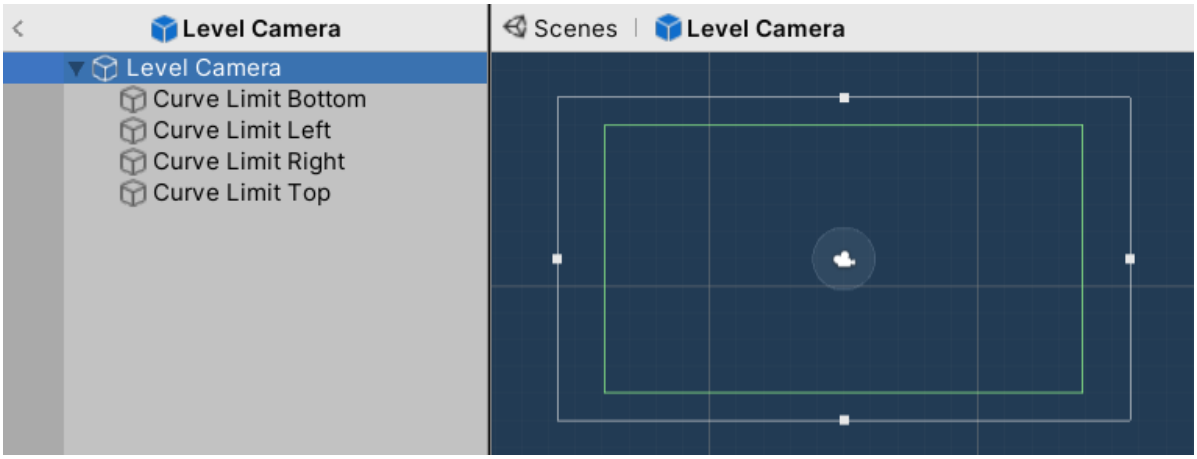
Là game object quản lý việc đếm số lần chết của Player trong từng màn chơi, cung cấp thông tin về số lần chết và phương thức để hồi sinh Player nếu cần.

```
public void ReviveMe(GameObject me, Vector2 pos, float time = 0)
{
  if (time > 0) StartCoroutine(ReviveLater(me, pos, time));
  else ReviveNow(me, pos);
}

private IEnumerator ReviveLater(GameObject me, Vector2 pos, float time)
{
  yield return new WaitForSeconds(time);
  ReviveNow(me, pos);
}

private void ReviveNow(GameObject me, Vector2 pos)
{
  me.transform.position = pos;
  me.SetActive(true);
}
```

Level Camera

Level Camera ở các Level Scene sẽ di chuyển theo game object Player, sử dụng tọa độ (x, y) của Player mỗi frame. Level Camera có size mặc định là 6.

Level Camera chứa 4 game object con có tag là CamEdge, là các Edge Collider 2D dạng trigger. Mục đích của các collider này là tạo vùng giới hạn cho đường curve được vẽ bởi Curve Renderer.



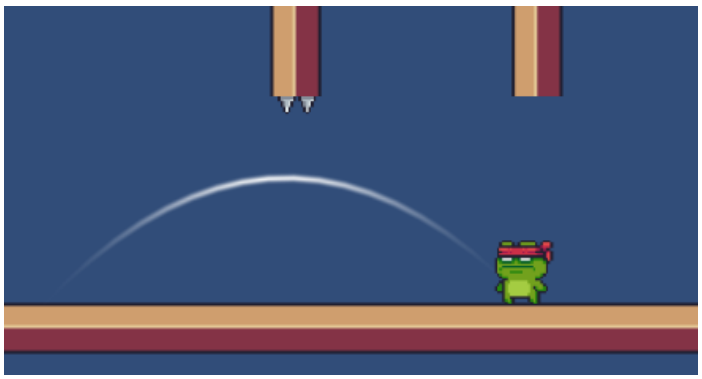
Arc Renderer

Arc Renderer là game object chịu trách nhiệm vẽ các arc lên màn hình nhằm hỗ trợ người chơi trong việc điều khiển Player. Arc là các cánh cung thể hiện vùng góc nhảy mà Player có thể nhảy. Tùy theo trạng thái của Player mà dạng arc phù hợp sẽ được vẽ. Arc chia thành hai dạng, dạng 1 là khi Player đang ở trạng thái đứng trên mặt đất - grounded (hình bên trái), dạng 2 là lúc Player đang bám tường - clinged (hình bên phải):

Arc sẽ được vẽ tại ngay vị trí mà người chơi chạm tay vào màn hình (trừ vị trí pause button) và tồn tại tới khi người chơi nhấc ngón tay khỏi màn hình. Arc Renderer sẽ tự động thay đổi dạng arc khi Player đổi trạng thái ngay cả khi người chơi chưa nhấc ngón khỏi màn hình vì phương thức RenderArc được gọi liên tục trong FixedUpdate.

Curve Renderer

Curve Renderer là game object làm nhiệm vụ vẽ curve (đường cong) thể hiện quỹ đạo nhảy của Player trên màn hình. Curve Renderer sử dụng component **Line Renderer** để vẽ curve vì về bản chất, curve được vẽ từ nhiều đoạn thẳng nhỏ.



Đầu tiên, Curve Renderer nhận vào một đối tượng Jump hợp lệ. Sau đó, dựa trên thuộc tính velocity và angle của jump đó, quỹ đạo nhảy của Player sẽ được tính toán. Các công thức dưới đây được tham khảo từ trang https://en.wikipedia.org/wiki/Projectile_motion. Để có thể biết được quỹ đạo mà không cần tới yếu tố thời gian t của lần nhảy, ta sử dụng công thức sau:

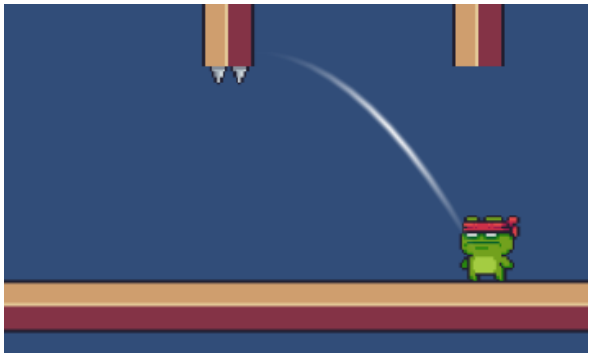
$$y = \tan(\theta) \cdot x - \frac{g}{2v_0^2 \cos^2 \theta} \cdot x^2$$

Tại những toạ độ x khác nhau, ta tìm được tọa độ y tương ứng của quỹ đạo nhảy. Tuy nhiên, với những lần nhảy thẳng đứng có góc nhảy 90 độ thì tan(90) không tồn tại nên ta không thể biết được độ lớn của y dù vận tốc ban đầu v có mang giá trị nào đi nữa. Vậy nên, đối với trường hợp Player nhảy thẳng đứng, ta sẽ sử dụng công thức sau:

$$h = \frac{v_0^2 \sin^2(\theta)}{2g}$$

Lúc này, h chính là y, và cũng là vị trí cao nhất mà Player có thể nhảy thẳng đứng với vận tốc ban đầu v mà không cần quan tâm tới x.

Điểm đầu tiên của curve có tọa độ x nằm ở giữa chân của Player, sau đó các đoạn thẳng nhỏ sẽ được tính toán lần lượt từ tọa độ x gần Player và ra xa dần. Tại mỗi lần một đoạn thẳng được tính xong, một Physic2D.**Linecast** cũng được tạo ra để kiểm tra xem đoạn thẳng vừa được tính có va chạm một collider nào không. Nếu có thì hàm số trả về tạo độ vị trí va chạm và việc tính toán curve được dừng lại. Vì curve sẽ chắc chắn phải va chạm một collider trên đường quỹ đạo của nó (ít nhất sẽ phải va chạm CamEdge của Level Camera) nên không có điểm dừng cho tạo độ x khi tính toán.



```
 private Vector3? GetRelativeHitPosition(Vector2 relativeStart, Vector2 relativeEnd,
Vector2 origin)
  {
    // hit every thing except the layer Curve Ignore
    int layerMask = ~LayerMask.GetMask("Curve Ignore");
    RaycastHit2D hit = Physics2D.Linecast(relativeStart + origin, relativeEnd +
origin, layerMask);
    if (hit.collider == null) return null;
    return hit.point - origin;
  }
```

Touch Detector

Đây là game object chịu trách nhiệm chuyển đổi các tap (chạm) của người chơi trên màn hình thành các lệnh điều khiển game object Player, cũng như làm cho các game object Curve Renderer và Arc Renderer thay đổi để phù hợp với tương tác của người chơi.

Để bắt được vị trí tap của người chơi trên màn hình điện thoại, các API của **UnityEngine.Input** được sử dụng. Touch Detector chỉ tiếp nhận một tap duy nhất, không xử lý đa chạm bằng việc liên tục kiểm tra Input.touchCount (1). Sau khi bắt được tap đầu tiên của người chơi, các phase của tap đó được tận dụng.

Trong TouchPhase.Began, ID của tap được kiểm tra với API **`IsPointerOverGameObject()`** của EventSystem, nếu người chơi đã chạm vào một Event Object (như pause button của HUD) thì tap đó không hợp lệ (2). Còn nếu hợp lệ, vị trị pixel của tap đó sẽ được lưu lại (3), đồng thời Arc Renderer (ar) cũng vẽ Arc lên màn hình (4).

Sau đó, nếu người chơi di chuyển ngón tay khi vẫn đang chạm màn hình thì chuyển sang TouchPhase.Moved, tại đây distance được tính toán bằng việc lấy vị trí hiện tại (5), chuyển đổi từ pixel về unit rồi trừ đi vị trị start ban đầu bằng phương thức **GetWorldDistance**. Sau đó, phương thức **GenerateJump** (xem mục Player) được sử dụng để tạo ra object jump, nếu jump hợp lệ (khác null) thì curve sẽ được vẽ (6).

Nếu người chơi tiếp tục nhấn giữ ngón tay không thả ra thì chuyển sang TouchPhase.Stationary. Đối tượng jump được override liên tục để giải quyết trường hợp khi Player đang di chuyển ở trạng thái Clinged thì curve cũng sẽ phải thay đổi theo (7).

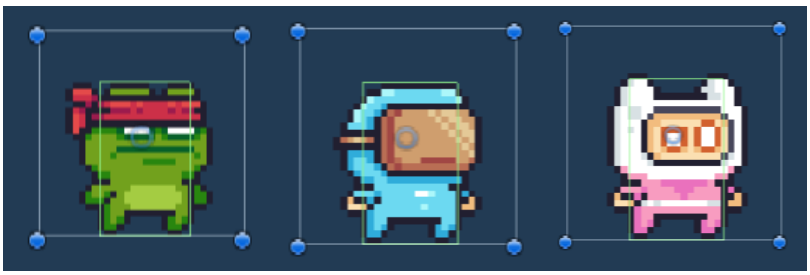
Khi người chơi nhấc ngón tay khỏi màn hình thì sẽ chuyển sang TouchPhase.Ended. Lúc này, nếu jump hợp lệ thì Player sẽ tiến hành phương thức **SetJump** để chuẩn bị nhảy (8).

```
  private void CheckUserTouch()
  {
    if (Input.touchCount != 1) return; (1)

    Touch touch = Input.touches[0];
    if (touch.phase == TouchPhase.Began)
    {
      if (EventSystem.current.IsPointerOverGameObject(touch.fingerId)) return; (2)
      start = Input.touches[0].position; (3)
```

```
      ar.SetOn(start); (4)
    }
    else if (touch.phase == TouchPhase.Moved)
    {
      Vector2 end = Input.touches[0].position; (5)
      distance = GetWorldDistance(start, end);

      jump = player.GenerateJump(distance);
      RenderCurve(); (6)
    }
    else if (touch.phase == TouchPhase.Stationary)
    {
      if (distance.Equals(Vector2.zero)) return;

      jump = player.GenerateJump(distance);
      RenderCurve(); (7)
    }
    else if (touch.phase == TouchPhase.Ended)
    {
      if (jump != null) player.SetJump(jump); (8)
      Reset();
    }
    else if (touch.phase == TouchPhase.Canceled)
    {
      Reset();
    }
  }


private Vector2 GetWorldDistance(Vector2 pixelStart, Vector2 pixelEnd)
  {
    Vector2 distance = Camera.main.ScreenToWorldPoint(pixelEnd) -
             Camera.main.ScreenToWorldPoint(pixelStart);

    // if the magnitude of the distance is too small or too big, clamp it according
             to the ratio
    if (distance.magnitude < minMagnitude)
    {
      distance *= minMagnitude / distance.magnitude;
    }
    else if (distance.magnitude > maxMagnitude)
    {
      distance *= maxMagnitude / distance.magnitude;
    }
    return distance;
  }
```

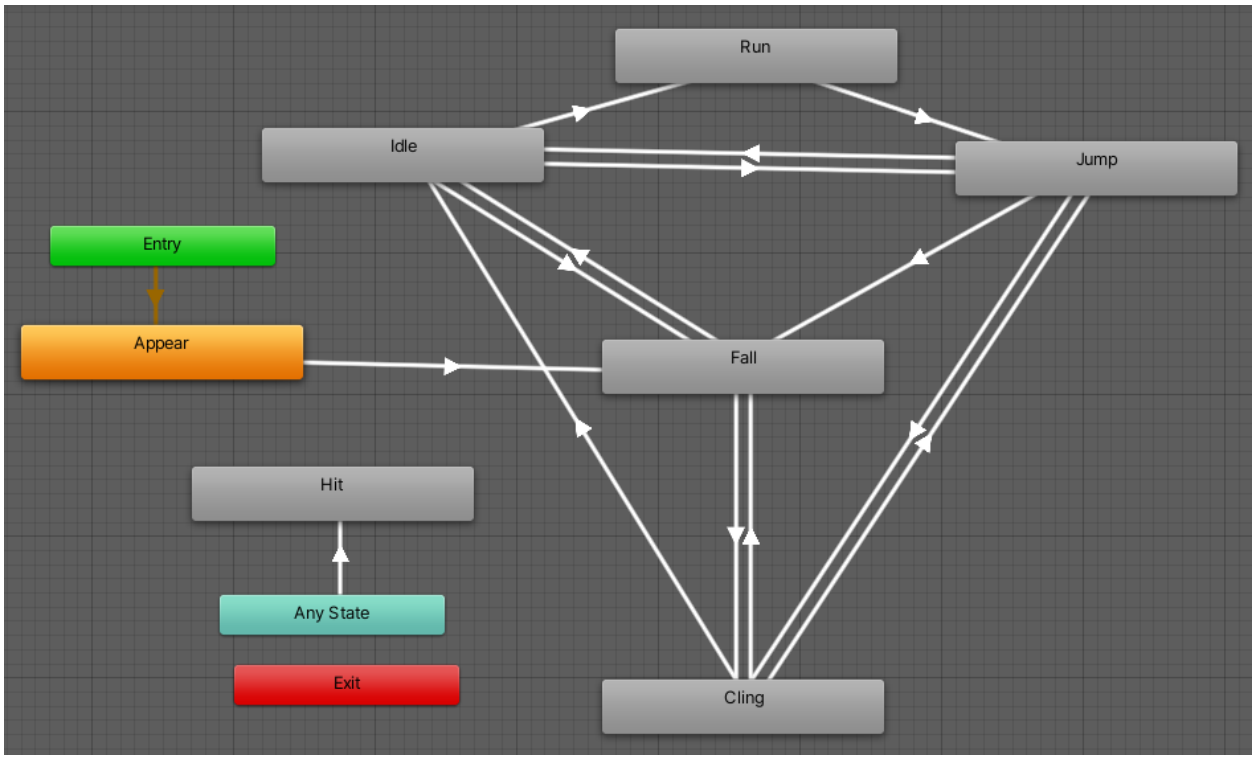
Player

Game object Player là object thể hiện nhân vật sẽ được điều khiển bởi người chơi. Hiện tại Pixel Jumper hỗ trợ 3 skin tương ứng với 3 prefab là Ninja Frog, Virtual Guy và Pink Man. Cả 3 prefab này tương tự nhau, chỉ khác ở các sprite được dùng trong animation.



Mỗi prefab được gắn 1 component Box Collider 2D, 1 component Animator, 1 component Script, 1 component Rigidbody 2D với Mass 1, Linear Drag 0, Angular Drag 0, Gravity Scale 3 và Collision Detection là **Continuous**.

Player có tất cả 7 animation clip, trong đó 6 clip gồm Appear, Fall, Idle, Jump, Cling và Hit là được dùng trong các Level Scene; riêng clip Run chỉ được sử dụng trong Menu Scene. Animator có 5 Parameter gồm Grounded (Bool), Clinged (Bool), Velocity Y (Int), Dead (Trigger) và Run (Bool). Animator sử dụng tất cả 14 transition để chuyển đổi state của Player.



Player script cũng cung cấp phương thức **GenerateJump** cho Touch Detector để lấy được đối tượng jump. GenerateJump nhận vào vector distance do Touch Detector cung cấp và tùy theo trạng thái và góc nhảy cho phép mà trả về một Jump object, object này sẽ null nếu vector distance không hợp lệ.

```
public Jump GenerateJump(Vector2 distance)
  {
```

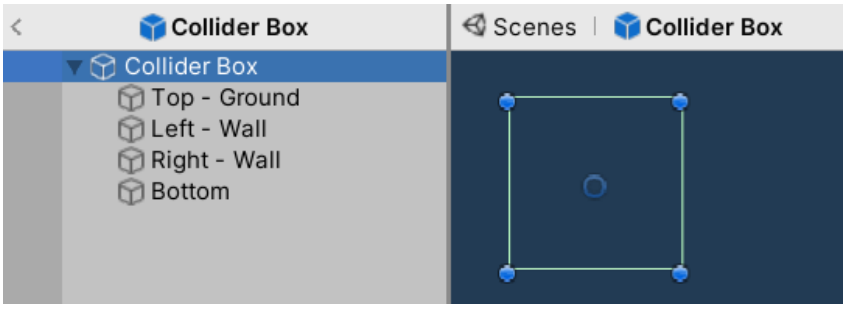
```
    float angle = CurveRenderer.CalculateAngle(distance);
    Vector2 velocity = distance * jvc; // jump velocity coefficient

    if (Grounded)
    {
      // vertical jump
      if (angle > 85 && angle < 95)
      {
        return new Jump(new Vector2(0, velocity.y), 90, JumpType.GROUND_VERTICAL);
      }
      else if (angle > 10 && angle < 170)
      {
        return new Jump(velocity, angle, JumpType.GROUND_NOT_VERTICAL);
      }
    }

    else if (Clinged)
    {
      // jump to the right
      if (transform.localScale.x < 0 && angle > -85 && angle < 65)
      {
        return new Jump(velocity, angle, JumpType.WALL_TO_RIGHT);
      }
      // jump to the left
      if (transform.localScale.x > 0 && (angle > 115 || angle < -95))
      {
        return new Jump(velocity, angle, JumpType.WALL_TO_LEFT);
      }
    }
    return null;
  }
```

Jump

Jump là class được tạo thêm để đóng gói các dữ liệu velocity, jump type và angle; mục đích là làm cho dữ liệu có cấu trúc, dễ đọc và thao tác hơn. Trong quá trình người chơi điều khiển nhân vật, các Jump object sẽ được trao đổi qua lại giữa các game object Touch Detector, Curve Renderer và Player.

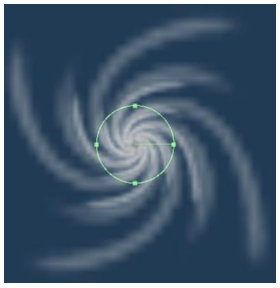
Collider Box

Collider Box là game object được dùng để làm các yếu tố vật cản như nền đất và tường bám trong game. Collider Box gồm 4 game object con, mỗi object được gắn một component Edge Collider 2D. Object ở vị trí top được gắn tag "Ground", 2 object ở vị trí trái và phải được gắn tag "Wall", object ở vị trí bottom không được gắn tag.



Exit

Exit là game object có hình dạng lốc xoáy, đại diện cho cổng Exit của một level. Khi người chơi điều khiển game object Player chạm vào vùng Circle Collider 2D ở tâm lốc xoáy thì phương thức **FinishLevel** của Game Controller được gọi. Lúc này, một Result Panel sẽ hiện ra thông báo kết quả cho người chơi, đồng thời số tiền và số lần chết cũng được lưu lại.



3.2 Data Flow

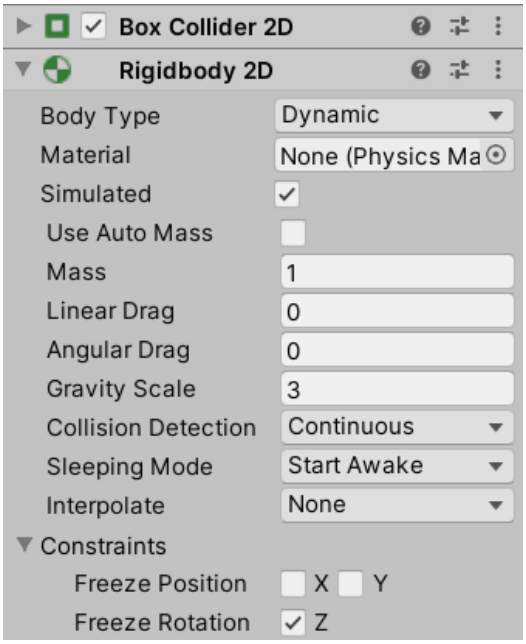
Khi người chơi mở ứng dụng, Game Controller sẽ thử load data được lưu trong playerPrefs (là Shared Preferences trong Android) từ file pixeljumper.data. Nếu file đó không tồn tại thì đây là lần đầu ứng dụng được mở.

Dữ liệu người chơi sẽ save xuống playerPrefs tại những thời điểm sau:

- Tắt mở audio tại Menu Panel hay Pause Panel.
- Thay đổi skin tại Level Panel.
- Unlock item mới tại Shop Scene.
- Hoàn thành một level (bằng cách chạm vào cổng Exit), lúc này số tiền và số lần chết mới được lưu lại.

3.3 Game Physics

Pixel Jumper sử dụng hệ thống Physics 2D được hỗ trợ trong Unity. Physics trong game chủ yếu xoay quanh tương tác giữa game object **Player** (do người chơi điều khiển) và các **Collider Box** (nền đất và tường bám). Game object Player được gắn thêm component Box Collider 2D dạng collision và Rigidbody 2D. Collider Box sử dụng 4 Edge Collider 2D dạng collision.



Tuy nhiên, việc Player tiếp tục trượt một khoảng nhỏ sau khi tiếp đất, điều này tuân theo cơ chế hoạt động của Rigidbody 2D, sẽ làm người chơi khó điều khiển được nhân vật hơn. Vì vậy, để Player có thể dừng lại ngay sau khi tiếp đất, thuộc tính velocity sẽ được thiết lập về 0 ngay khi Player chạm mặt Ground của Collider Box.

Khi Player chạm vào mặt Wall của Collider Box, Player sẽ trượt (rơi) rất chậm xuống dưới. Để đạt được điều này, thuộc tính drag (Linear Drag) của Rigidbody sẽ được thiết lập lên 50 ngay khi Player chạm vào mặt Wall. Nếu Player thoát khỏi mặt Wall của Collider Box, drag sẽ trở về 0.

Ngoài ra, khi Player chết, để Player đứng yên trong không trung khi animation Dead đang chạy, thuộc tính isKinematic của Rigidbody được chuyển về true. Thuộc tính đó sẽ được chuyển về false khi Player được hồi sinh.

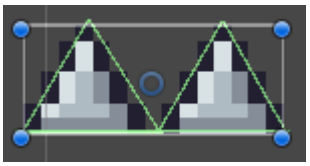
3.4 Level Object Details

Trong pixeljumper hệ thống chướng ngại vật bao gồm 2 phần chính là Monster và Trap. Có nhiệm vụ là ngăn chặn người chơi tiếp cận “exit” yếu tố chính giúp người chơi vượt Lever và “coins” yếu tố phụ, giúp người chơi sưu tập “skin” như đã đề cập ở trên.

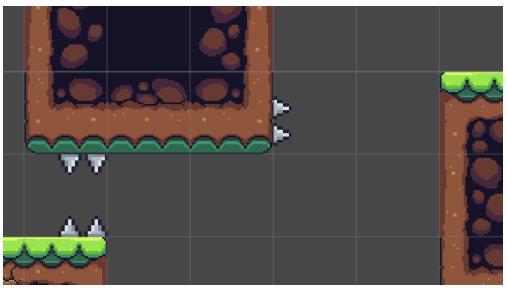
Nhằm mục đích tăng độ khó và tăng tính hấp dẫn cho trò chơi. Giúp pixeljumper phong phú về nội dung mà vẫn giữ được phong cách chơi đơn giản như hầu hết game mobile hiện nay.

3.4.1 Trap

Gameobject Trap thể hiện bằng sprite bên dưới.



Được để ở “mặt đất” hoặc trên “tường” nhằm mục đích cản trở “player” di chuyển(nhảy hoặc trượt trên tường).

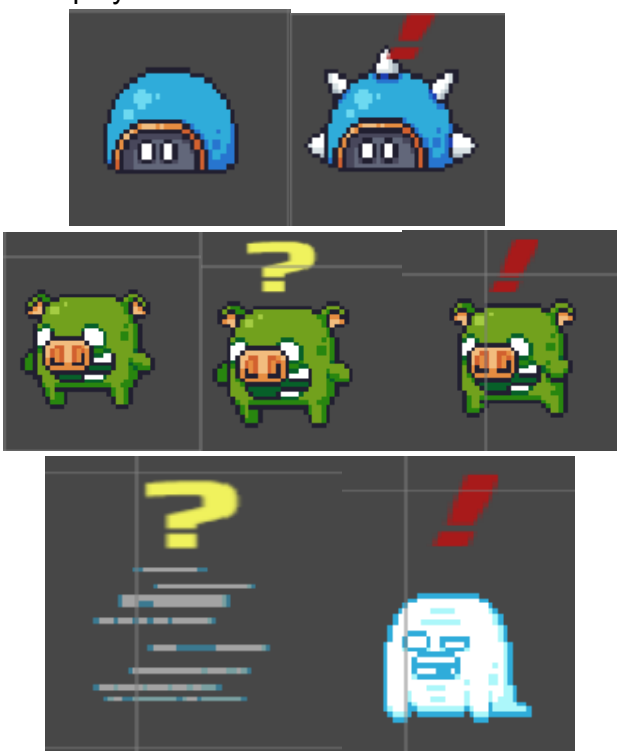


Gameobject Trap chỉ gồm 1 component chính là Edge Collider 2D để tận dụng phương thức OnColliderEnter2D() với tag là “Spike”.

3.4.2 Monster

Khác với trap chỉ đặt cố định, áp đặt “player” đi theo lối đi được gợi ý sẵn cũng như không có tương tác, phản ứng với “player”. Các Gameobject monster có các:

Phản ứng khác nhau đối với “player”.



“Chướng ngại” tạo ra “chướng ngại”

Hoặc di chuyển thật chậm trên mặt đất.

Đòi hỏi người chơi không những khéo tay khi di chuyển trên các bờ tường tránh những bẫy đặt sẵn, mà còn tính toán đối với từng Monster.

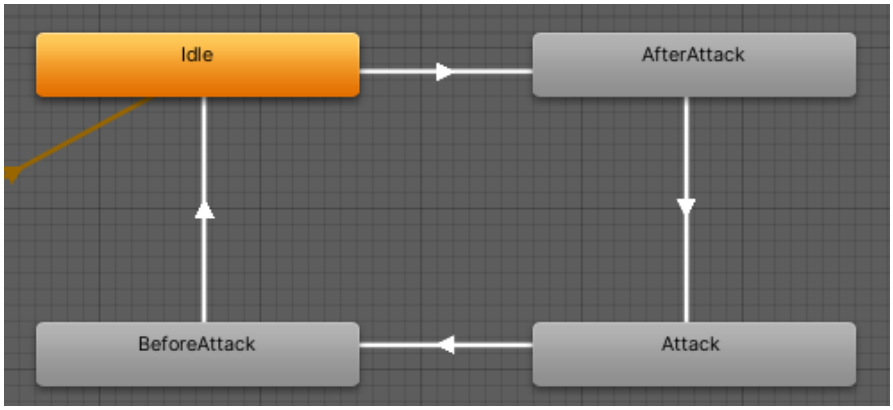
3.4.2 Monster AI

Turtle

Là một monster không thể duy chuyển. “Turtle” sẽ có hoạt ảnh giương gai khi người chơi lại gần và thu gai lại khi “player” di chuyển đi đủ xa.

“Turtle” chỉ gồm 1 component được thêm vào là CircleCollider2D.

4 trạng thái của Turtle”



4 clips và chỉ 4 Transition, với 2 clip chính là Idle và Attack.
Hành vi “Turtle” hoạt động dựa trên khoảng cách của “player” và bản thân “Turtle”

```
Vector2 Posi1, Posi2;
Posi1 = transform.position;
Posi2 = playerTran.position;
double temp = Math.Round(((Posi1.x - Posi2.x) * (Posi1.x - Posi2.x) + (Posi1.y - Posi2.y) * (Posi1.y - Posi2.y)), 1);
rangeToPlayerx2 = float.Parse(temp.ToString());
```

Nếu khoảng cách thoả yêu cầu thì set hoạt động tương ứng.

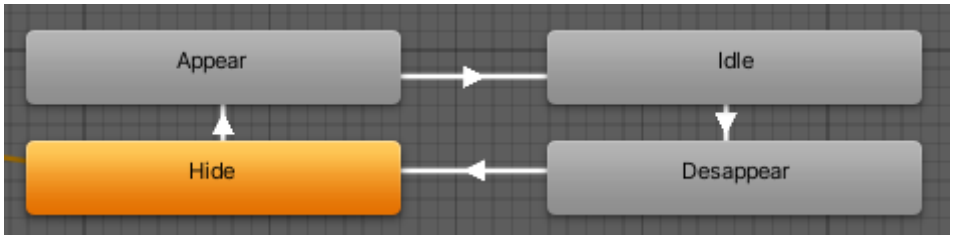
```
if (rangeToPlayerx2 < 40 && !animator.GetBool("Attacking"))
{
    animator.SetBool("AfterAttack", true);
    chamThan.gameObject.SetActive(true);

}
else if(rangeToPlayerx2 >40 && animator.GetBool("Attacking"))
{
    animator.SetBool("BeforeAttack", true);
}
```

Ghost

Là một Monster không thể di chuyển tương tự như “Turtle”, sẽ hoạt ảnh hiện ra nếu “player” di chuyển lại gần và hoạt ảnh ẩn đi (mờ mờ vẫn có thể thấy) nến “player” đi chuyển đủ xa. Điểm khác biệt còn ở chỗ “Ghost” được đặt không chỉ trên mặt đất mà còn giữa không trung.

“Ghost” chỉ gồm 1 component được thêm vào là CircleCollider2D.

4 trạng thái của “Ghost”



4 clips và chỉ 4 Transition, với 2 clip chính là Hide và Idle.
Hành vi của “Ghost” hoạt động tương tự “Turtle”.

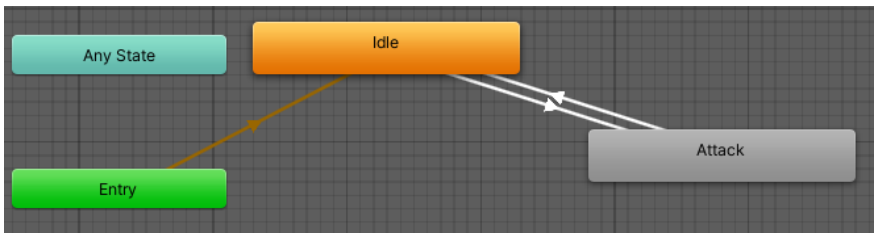
3. Plant

Là một Monster không thể di chuyển nhưng có thể tấn công người chơi một cách không chủ động. “Plant” được đặt trên mặt đất và phun ra hạt đậu mỗi vài giây. Gameobject plant được kèm theo “Fire” và “Limit”.

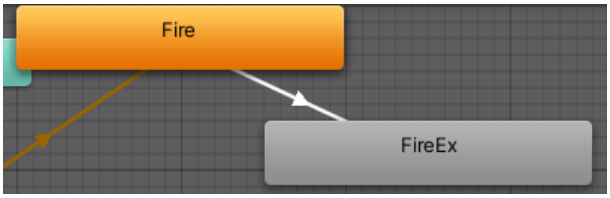
➔ “Plant” chỉ gồm 1 component được thêm vào là BoxCollider2D.

➔ “Fire” gồm 2 component là Rigidbody2D và BoxCollider2D. “Fire” được kèm theo gồm “Fire1” hình hạt đậu, và “Fire2” hoạt ảnh ngọn lửa.

➔ “Limit” chỉ gồm 1 component được thêm vào là BoxCollider2D. Là điểm giới hạn của “Fire” có thể bay tới.



2 trạng thái của plant



2 trạng thái của fire2

Hành vi của “Plant” đơn giản,là liên tiếp tạo ra “Fire” trong khoảng thời gian cố định.

```
Vector2 v3 = transform.position;
if (transform.localScale.x > 0)
{
    v3.x += -0.2129382f;
    v3.y += 0.03266411f;
    Instantiate(FireL, v3, Quaternion.identity);
}
else
{
    v3.x -= -0.132f;
    v3.y -= -0.058f;
    Instantiate(FireR, v3, Quaternion.identity);
}
```

4. User Interface

4.1 Game Shell

4.2 Game Screens

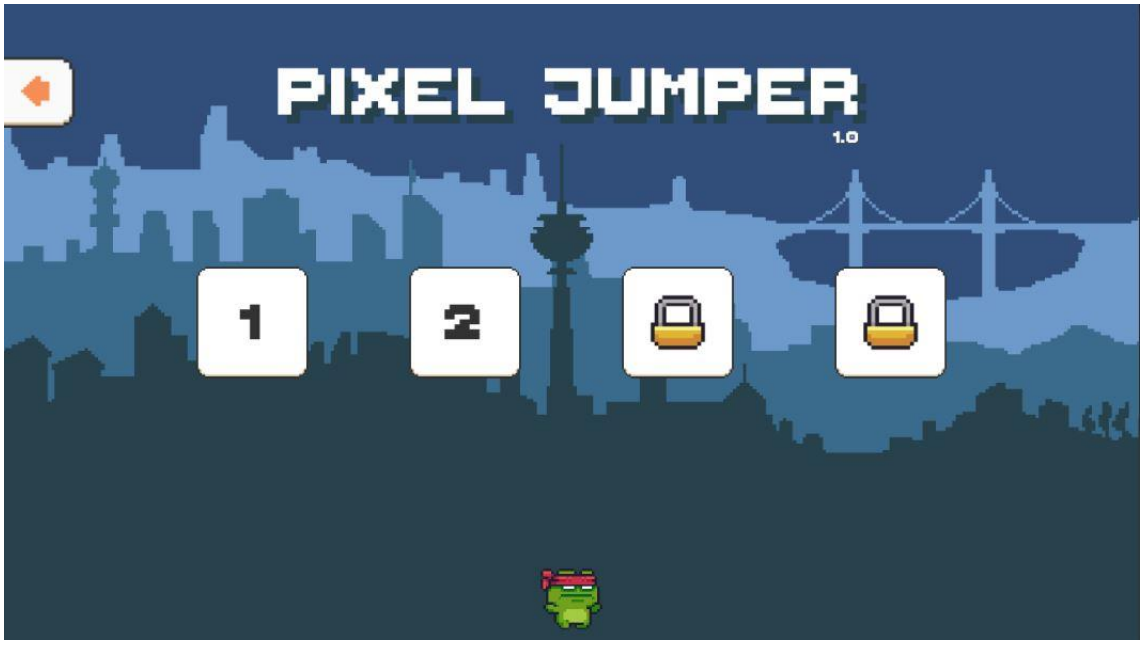
4.2.1 Menu Panel

Người chơi sẽ nhìn thấy Menu Panel đầu tiên khi mở ứng dụng Pixel Jumper. Tại đây, người chơi có thể nhấn PLAY để đi tới Level Panel, nhấn SHOP để tới Shop Scene hoặc tắt mở audio bằng button ở góc dưới trái màn hình. Nếu người chơi chưa unlock skin mới thì sẽ chỉ có một game object Player mặc định là Ninja Frog xuất hiện.

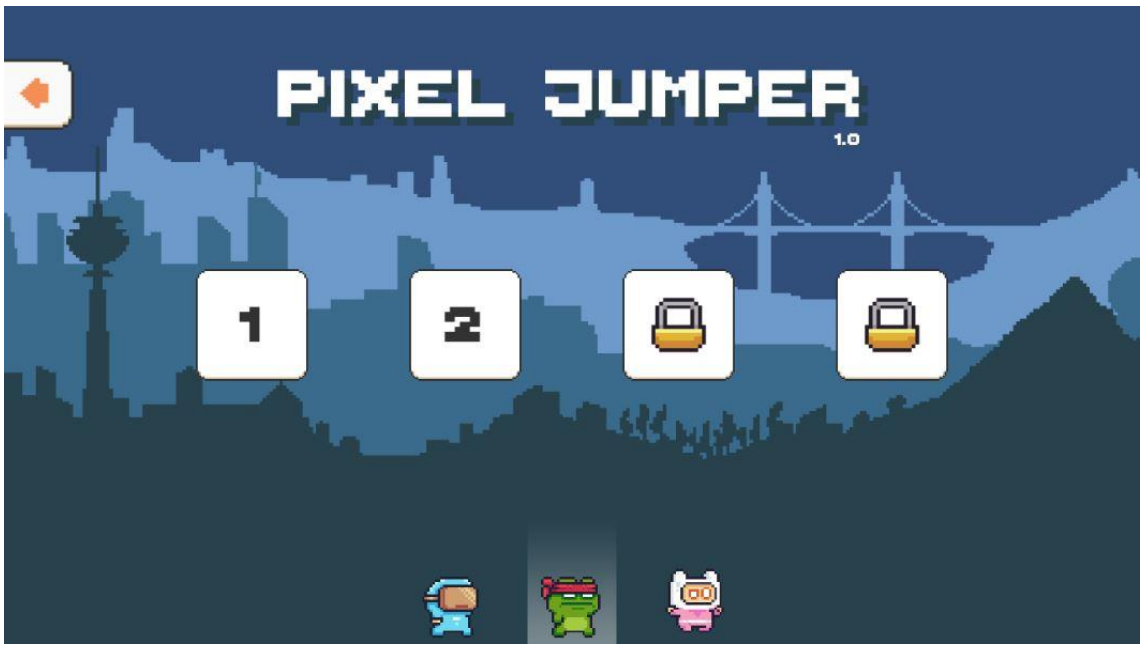
Nếu người chơi đã unlock skin mới từ Shop Scene thì sẽ có thêm các game object Player khác xuất hiện tùy theo skin nào đã được unlock.

4.2.2. Level Panel

Là panel để người chơi lựa chọn màn chơi. Level tiếp theo chỉ được mở khi người chơi tới được cổng Exit của level trước đó. Hiện tại, Pixel Jumper hỗ trợ 2 level. Người chơi có thể nhấn nút mũi trên ở góc trên trái để trở về Menu Panel.

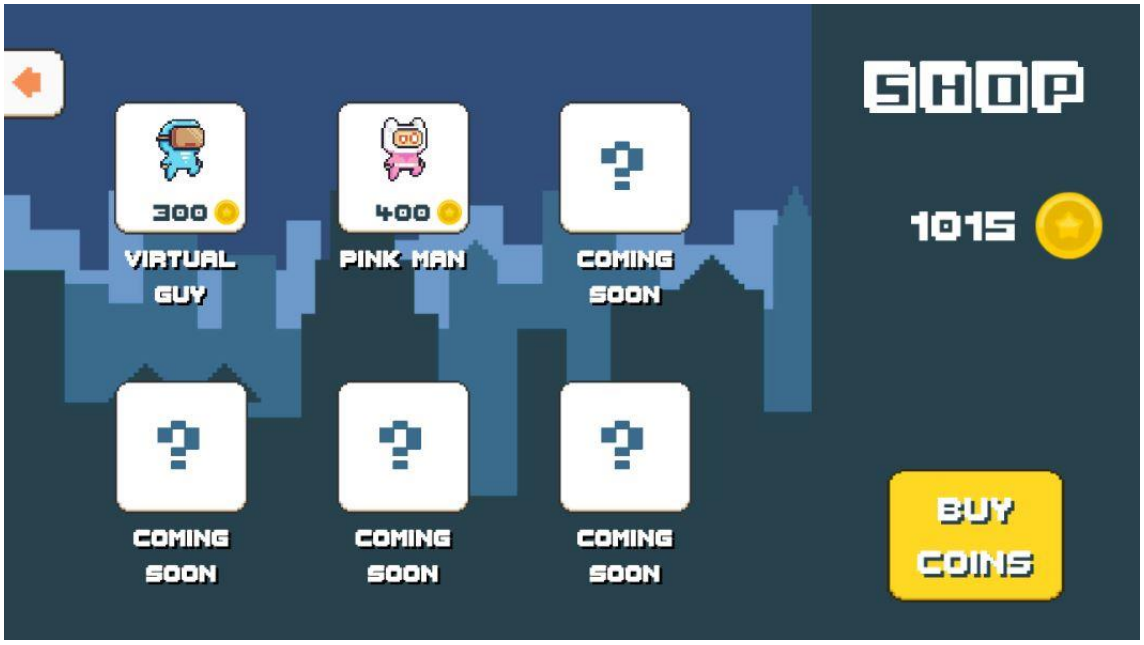


Nếu người chơi đã unlock thêm một skin bất kỳ từ Shop Scene thì chức năng thay đổi skin sẽ xuất hiện (chính là game object Selector). Người chơi có thể chạm vào skin mình muốn và một vệt sáng sẽ xuất hiện phía sau skin đó để thể hiện skin đã được chọn.

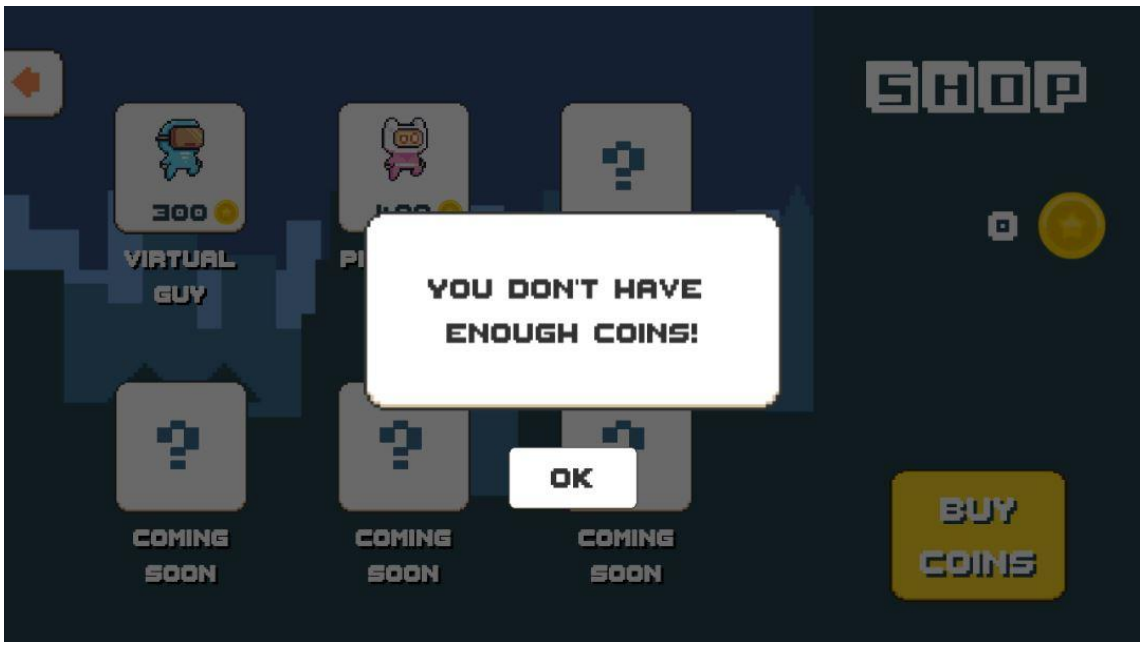


4.2.3 Shop Scene

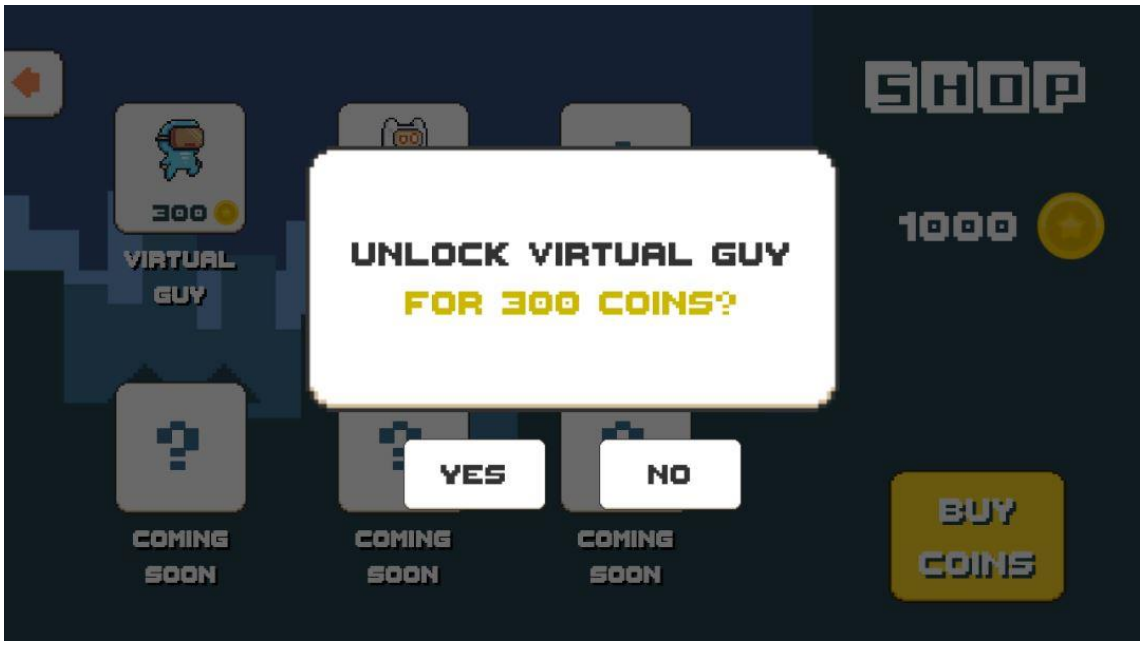
Shop Scene là nơi người chơi có thể unlock item mới, hiện tại người chơi Pixel Jumper có thể mở thêm 2 skin mới là Virtual Guy và Pink Man. Người chơi có thể dùng nút mũi tên ở góc trên trái của màn hình để trở về Menu Panel.



Nếu người chơi không đủ tiền để unlock một item thì thông báo sẽ xuất hiện:



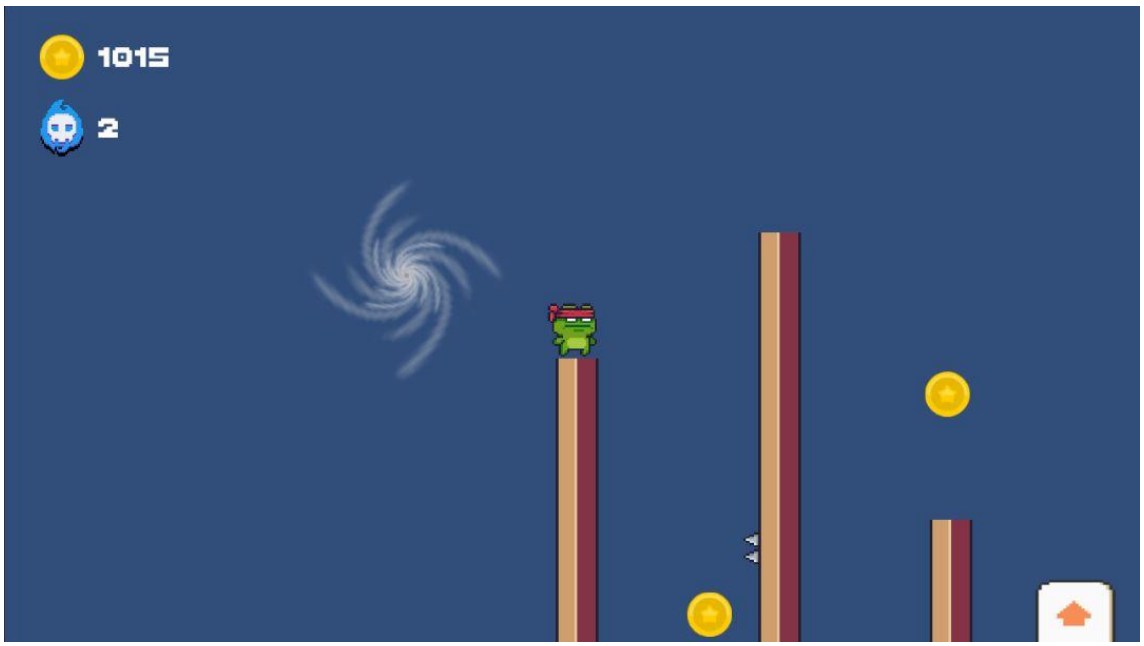
Ngược lại, nếu người chơi đủ tiền để unlock item thì một panel sẽ xuất hiện để xác nhận:



4.2.4 Level 1 (Demo Level)

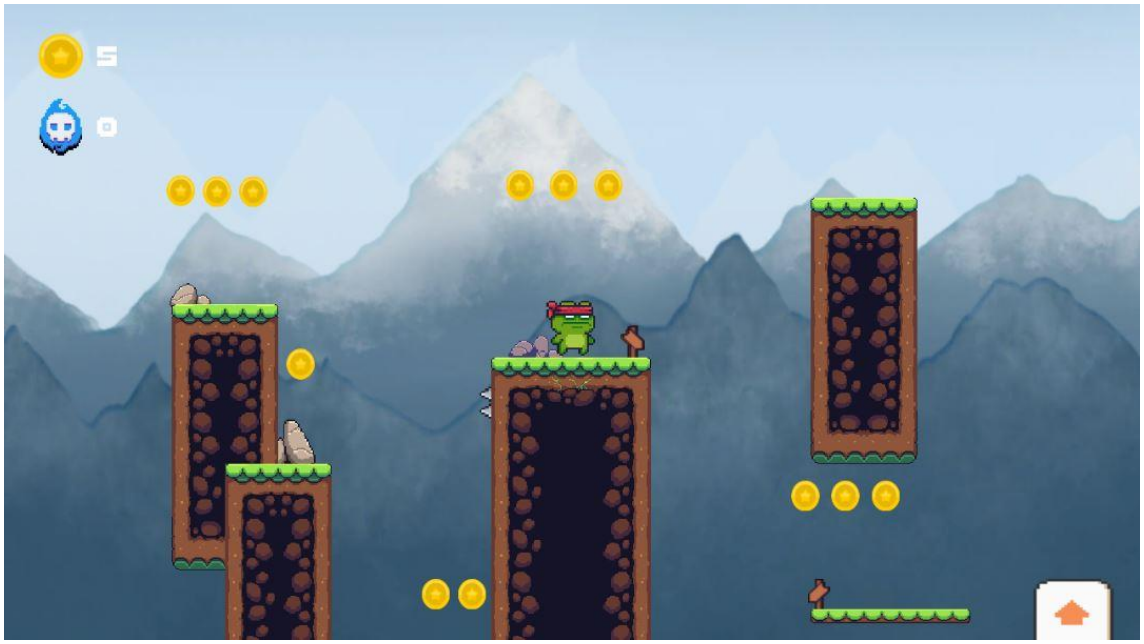
Đây là level đầu tiên của game, nhưng thực chất đây chỉ là một test level được nhóm dùng để thử nghiệm trong quá trình phát triển game. Test level được đưa vào game với mục đích chính là để demo nhanh các chức năng của Pixel Jumper trong giai đoạn nhóm báo cáo dự án. Do đó, level này sẽ bị gỡ bỏ khi sản phẩm đã hoàn thiện.

Để giúp người chơi có tầm quan sát map tốt hơn, HUD của game được giữ ở mức đơn giản nhất có thể. Phía trên trái của màn hình là số tiền đã thu thập được và số lần Player đã chết. Phía dưới phải của màn hình là pause button, khi nhấn chọn sẽ mở ra Pause Panel. Ngoài ra, Player bắt buộc phải chạm vào tâm lốc xoáy để thắng màn chơi, chỉ khi đó thì số tiền đã thu thập mới được lưu và level tiếp theo mới được mở.



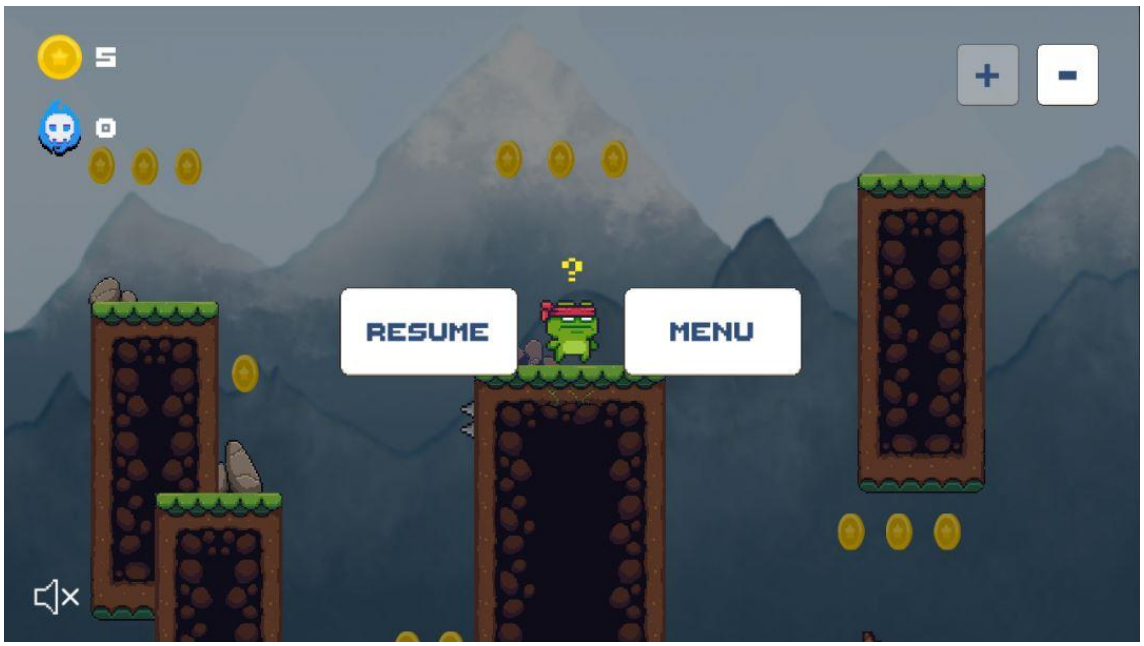
4.2.5 Level 2

Đây là level tiếp theo của game sau khi người chơi đã hoàn thành Level 1. Map của level này được xây dựng thông qua hệ thống vẽ Tilemap của Unity, nhờ vậy mà quá trình tạo map diễn ra hiệu quả hơn hẳn. Người chơi cũng sẽ gặp nhiều loại Enemy khác nhau trong level này.



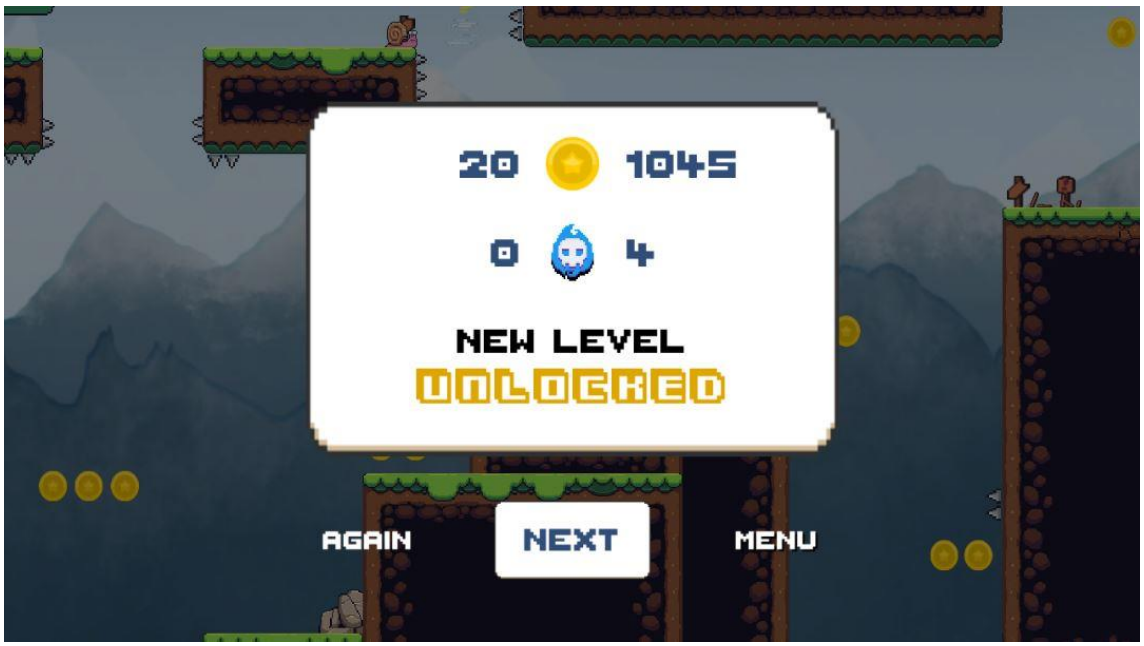
4.2.6 Pause Panel

Pause Panel được mở khi người chơi nhấn nút pause ở góc dưới phải màn hình. Lúc này, toàn bộ game được dừng lại. Người chơi sẽ có các tùy chọn settings khác nhau như nút tắt mở audio ở góc dưới trái, 2 nút phóng to và thu nhỏ view nhìn ở góc trên phải, nút tiếp tục game hay nút thoát level và trở về Menu Panel nằm ở giữa.

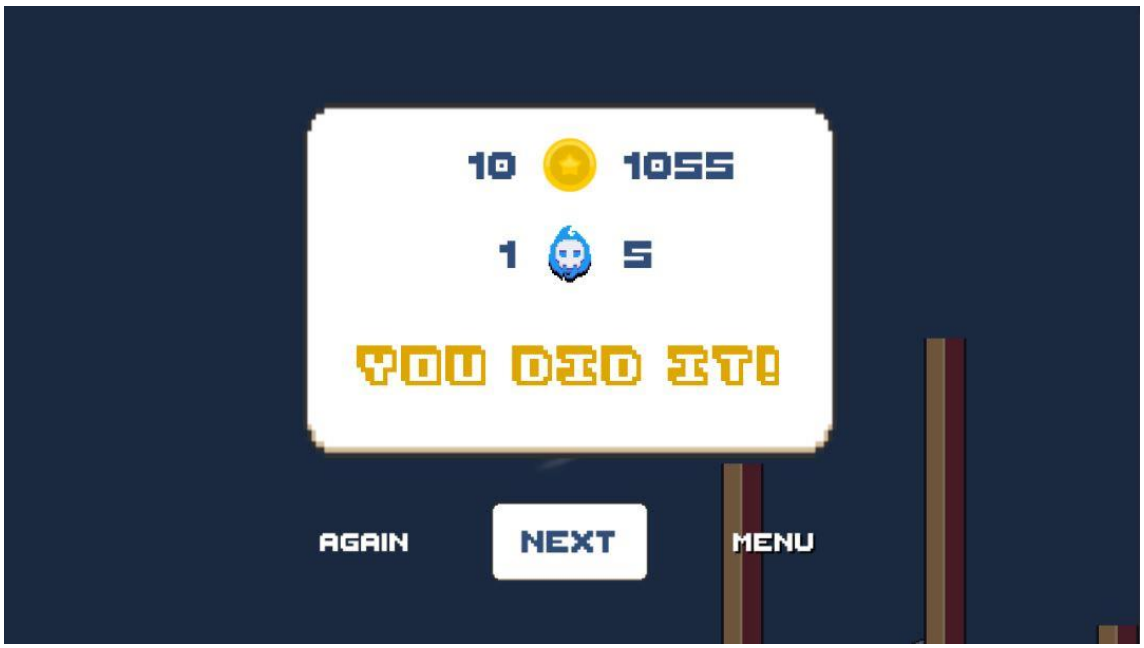


4.2.7 Result Panel

Result Panel sẽ hiện ra khi người chơi đã hoàn thành level bằng cách điều khiển Player chạm vào tâm của lốc xoáy. Hai dòng đầu tiên là số tiền và số lần chết của Player, bên trái là số liệu của level vừa hoàn thành, bên phải là con số tổng của tất cả level. Nếu đây là lần đầu tiên người chơi hoàn thành level thì dòng chữ "New level unlocked" sẽ hiện ra để thông báo rằng level tiếp theo đã được mở.



Người chơi có thể chọn AGAIN để chơi lại level, chọn NEXT để sang thẳng level tiếp theo hoặc chọn MENU để trở về Menu Panel. Nếu người chơi hoàn thành lại một level đã được unlock trước đó thì dòng chữ "You did it!" sẽ hiện ra:


10
1055
1
5
YOU DID IT!
AGAIN
NEXT
MENU

5. Art and Video

5.1 Graphic Engine

- Unity Tilemap: cung cấp nhiều công cụ để thiết kế nhanh các tilemap.
- Pixlr.com: là trình chỉnh sửa hình ảnh online, dùng để tinh chỉnh một số hình ảnh và background của game.

5.2 Introduction to Artists

Game Pixel Jumper có sử dụng nhiều Unity asset và các resource miễn phí từ Internet:

- Vortex exit - planetclicker-assets - FacadeGaikan - Opengameart.org.
- Sprite pack #1 - tap and fly - render knight - Unity Asset Store.
- Free pixel font - thaleah - Unity Asset Store.
- Pixel adventure 1 & 2 - pixel frog - Unity Asset Store.

6. Sound and Music

Game Pixel Jumper có sử dụng nhiều audio miễn phí từ nhiều nguồn trên Internet để làm background music và SFX:

- Menu bgMusic - dance and jump loop - snabisch - Opengameart.org
- Shop bgMusic - item shop - controllerhead - Opengameart.org
- Level 1 bgMusic - jump and run - tropical mix - bart - Opengameart.org
- Level 2 bgMusic - jump and run (8-bit) - tropical mix - bart - Opengameart.org
- Collide SFX - jump landing sound - MentalSanityOff - Opengameart.org
- Sound FX - retro pack - zero rare - Unity Asset Store.