
EPTA ENTERTAINMENT

NESTE DOCUMENTO ENCONTRA-SE O PLANO DE DESENVOLVIMENTO QUE SE REFERE À CRIAÇÃO DA VERSÃO 1.0 DO APLICATIVO EPTA SPACE PROGRAM, DESENVOLVIDO PELA EPTA ENTERTAINMENT.

DESENVOLVIDO POR

FELIPE J. O. RIBEIRO
MATEUS DA SILVA FERNANDES
OLAVO CAETANO INÁCIO
PEDRO GUILHERME R. V. DE MELO

*Universidade Federal de
Uberlândia*

EQUIPE DE PROPULSÃO E TECNOLOGIA
AEROESPACIAL

EPTA SPACE PROGRAM - VERSÃO 1.0

19 DE JANEIRO DE 2021

Resumo

Neste documento encontram-se as práticas adotadas no desenvolvimento da versão 1.0 do aplicativo Epta Space Program. O programa já havia sido lançado na google play em meados de 2017, mas ele foi retirado do ar devido a questões burocráticas (ausência de uma política de privacidade).

Nesta etapa de desenvolvimento a equipe atualizará o jogo para a versão mais recente do unity. Também revisará o work flow uma vez que na versão de 2017 estava impossível de escalar e inviável de manter devido a más práticas de programação. Como resposta a isso, a sub área se propôs a revisar todos os padrões de desenvolvimento de forma a estar conforme com as mais altas recomendações desta indústria.

As funções base do aplicativo são:

- **TELA INICIAL** Desenvolvimento da tela inicial com animações. Que não precise iniciar o jogo pausado.
- **3 FASES** Três etapas distintas de jogo. (céu azul, extratosfera, espaço)
- **FASE FINAL** A certa altura (a determinar), o player tem a opção de ir para a lua. (o conteúdo será adicionado posteriormente. Mas por hora o player recebe uma mensagem de congratulações)
- **MORTE** Som e efeito visual de explosão no momento de morte.
- **ARCADE** Mecânica base do jogo mantém-se arcade, com rolagem para cima com obstáculos e replay ágil.

As features que espera-se que estejam presentes no jogo na data de estreia são:

- **PROPAGANDA** As propagandas tipo banner e tipo Intersticial devem ser implementadas novamente.
- **VERSÃO PAGA** Deve haver uma versão sem propagandas e que é paga. (preço a determinar)
- **TABELA INFO** Tabela de informações sobre a equipe no menu opções.
- **VOLUME** Deve ser possível ajustar o volume.
- **SALVAR** A melhor pontuação do jogador deve ser salva localmente, assim como o volume.
- **ASSETS REUTILIZADOS** Serão utilizados os assets do antigo lançamento. Novos devem ser feitos conforme necessidade.
- **ALTURAS REAIS** As alturas de mudança de fase devem ser consistentes com o mundo real.
- **DIFICULDADE CRESCENTE** O sistema de dificuldade deve ser implementado de forma a permitir razoável controle em função de fase e tempo.

A metodologia adotada para a criação do APP tem duas etapas importantes: Até o término da versão 1.0 será adotada uma metodologia em cascata, nesta que é a etapa de pré lançamento, onde se dividirão as features a implementar em uma ordem lógica de execução. Datas para pontos chave no desenvolvimento serão determinadas seguindo o padrão industrial (pre-alpha : 0.5.0, alpha : 0.6.0, beta-1 : 0.7.0, beta-2 : 0.8.0, ..., release : 1.0.0), onde cada uma terá um objetivo e uma lista de requerimentos a serem atendidos. Em seguida, após lançado, será adotada a metodologia ágil para desenvolvimento contínuo de atualizações e melhoramentos conforme necessário. Esse será chamada de etapa contínua de pós lançamento.

Será implementado um ciclo de desenvolvimento completamente integrado à ferramenta Git de controle de versão e à página GitHub (https://github.com/Epta-space/EPTA_SPACE_PROGRAM), onde o jogo será salvo em cloud em um repositório privado. Assim, essas ferramentas servirão para possibilitar o desenvolvimento em grupo do código fonte, além de também ser uma forma de quantificação da produtividade, uma vez que o controle do número de commits é uma métrica importante no acompanhamento das atividades. Mais detalhes podem ser encontrados na descrição das etapas de desenvolvimento do pré lançamento.

A criação do produto mínimo viável já foi feita na primeira versão do jogo, motivo pelo qual não temos a etapa MVP neste texto. Apesar disso será inserida aqui alguma documentação sobre esse processo nos capítulos iniciais. O desenvolvimentos que ocorreram anteriores a esse relatório são o motivo, também, da enumeração das etapas começarem em 0.5. Nos capítulos iniciais também será dada uma detalhada noção do estado atual do aplicativo assim como algumas informações sobre como foi o desenvolvimento até então. Tudo que foi feito até o momento foi desenvolvido durante o ano corrido de 2020, com exceção dos assets que vem da versão anterior.

Dentre as informações do estado atual do aplicativo serão listados os scripts que existem atualmente em formato UML e por escrito, a estrutura de arquivos no projeto unity e a estrutura de Prefabs. Estas estruturas também já atendem a padrões de projeto sugeridos online e facilitam o desenvolvimento do controle de versão via Git, além de organizar as informações para facilitar o acesso e aumentar a escalabilidade.

Próximo ao fim do documento também serão apresentadas algumas propostas de melhoramento na etapa ágil de lançamento, isso é, após a primeira versão do programa ser lançada na Google Play. Isso mostra as possíveis rotas que a subárea tomará no futuro, após o começo da etapa de desenvolvimento contínuo.

Palavras chave: Desenvolvimento de Software, aplicativo, android, jogo, google play.

Sumário

1	O desenvolvimento até aqui	1
1.1	Sobre o papel do desenvolvimento de jogos na equipe	2
1.2	O Mínimo produto viável(MVP)	2
2	Estado do aplicativo	3
2.1	Disposição de arquivos	3
2.2	Disposição de Game Objects	4
2.3	Ecossistema lógico	6
3	Plano de gestão para 2021	13
4	Planos futuros	15

Lista de Figuras

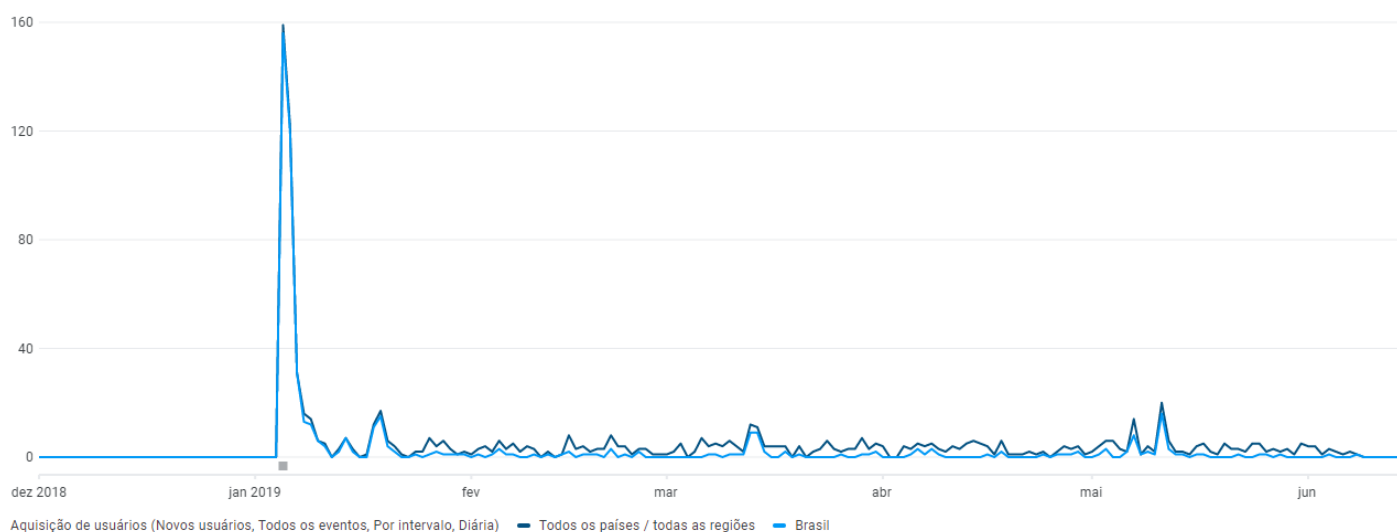
1.1	Downloads do aplicativo distribuídos no tempo.	1
2.1	Sistema de arquivos da pasta Assets. O que fora ignorado não tem importância para o desenvolvimento atual.	4
2.2	Árvore de objetos de jogo, prefabs em azul.	5
2.3	Código do script "Destroy_obstacles.cs".	6
2.4	Código do script "move.cs".	7
2.5	Código do script "interface_handler.cs".	8
2.6	Código do script "movimentação.cs".	9
2.7	Código do script "Object_Selector.cs".	10
2.8	Código do script "Game_Manager.cs".	11
2.9	Código do script "audio_controls.cs".	12

Capítulo 1

O desenvolvimento até aqui

O início deste projeto ocorreu no começo do primeiro semestre de 2018. Um grupo de membros se juntou no desenvolvimento do aplicativo como uma nova proposta de angariação de fundos para a equipe. Observou-se o potencial do mercado de aplicativos global, que cresce consistentemente ao longo dos anos. Além disso pensou-se na oportunidade de se aprofundar no desenvolvimento de software e o aprendizado que isso traria.

Ao longo do ano o aplicativo foi desenvolvido e finalmente lançado em dezembro de 2018. Ele foi baixado em mais de 40 países e mais de 800 vezes, com uma avaliação média de 4.88 estrelas de 5 (Fig. 1.1):



Em julho de 2019 o aplicativo foi retirado da Play Store devido à ausência de uma política de privacidade. Isso deu início a um longo hiato no desenvolvimento que só foi retomado no início de 2020.

A estratégia de monetização usada foi a de mostrar propaganda a partir dos serviço AdMob da Google. No tempo que o aplicativo ficou disponível para download (5 meses), somente com publicidade exibida durante as partidas, foi possível angariar 14 dólares. Não foi possível resgatar o valor, uma vez que só se pode fazer isso em pacotes de 100 dólares.

Considerando que o aplicativo só ficou disponível por 5 meses e que estava em seu estado base, sem qualquer atualização ou melhoramento, considera-se este retorno um ótimo resultado. Espera-se ultrapassar essa marca com o novo lançamento em 2021.

1.1 Sobre o papel do desenvolvimento de jogos na equipe

Este é um assunto de grande complexidade.

Na ciência da computação, o assunto é tratado como um dos mais difíceis, uma vez que, assim como na criação de foguetes, é uma prática extremamente multidisciplinar. Dentre as competências que se espera de uma equipe que se propõe a tal papel temos todo tipo de habilidade que oscila entre técnico e artístico de forma dramática.

Tendo a diversidade do assunto em vista, observa-se uma oportunidade de uma rica interlocução de agentes das mais diversas áreas, assim como vemos de um ponto de vista geral, dentro da equipe de propulsão e tecnologia aeroespacial. Desse forma espera-se que a subárea de desenvolvimento de jogos usufrua dessa diversidade que existe dentro da equipe, e até a expanda ainda mais.

Outro aspecto importante desta prática é a forma como se enxerga a programação. Na pesquisa e na engenharia também é constante a prática de programar, mas isso é feito de forma pragmática e apressada. Esse é o natural, uma vez que o código não é o produto final, mas um meio que se precisa percorrer, seja para uma pesquisa ou para desenvolver uma estrutura ou peça para o foguete. Tudo isso é familiar aos outros integrantes da equipe e para qualquer um da área de engenharia. Mas, dentro da EPTA Entertainment a perspectiva é outra. O código que está sendo desenvolvido é o produto final, e como tal, deve ser feito de forma bem diferente do código acadêmico. Além de funcionar ele deve ser legível e obedecer a padrões de projeto. Além disso, no lugar de exercer uma funcionalidade única e específica, códigos como os desenvolvidos na subárea muitas vezes precisam ser flexíveis e robustos de forma a trabalharem harmonicamente no ecossistema lógico do aplicativo.

Desenvolver essa outra perspectiva de desenvolvimento de software é extremamente positivo a qualquer engenheiro, uma vez que mais e mais o engenheiro moderno se vê cercado por programação, em qualquer especialidade.

1.2 O Mínimo produto viável(MVP)

O jogo para celular "Epta space program" é um programa desenvolvido para Android cujo público alvo são os entusiastas da indústria aeroespacial. Desse forma, o objetivo da equipe é criar algo acessível e informativo.

Capítulo 2

Estado do aplicativo

O aplicativo já se encontra em um estado avançado de desenvolvimento, consequência dos anos de trabalho que se foram. Nesse capítulo será descrito com ele se encontra no início de 2021.

Dentre as coisas descritas temos os seguintes tópicos:

- Disposição dos arquivos.
- Disposição de Game Objects.
- Ecossistema lógico.

2.1 Disposição de arquivos

O gerenciamento de arquivos é muito importante no desenvolvimento de jogos.

O objetivo é organizar a informação, uma vez que existem muitos arquivos (centenas). E podem ser figuras, música, scripts, meta dados, e objetos lógicos de jogo, como prefabs, animações, cenas e muitas outras coisas. Dessa forma é muito importante que tais arquivos sejam dispostos de uma forma inteligível, uma vez que mexer neles faz parte do cotidiano dos desenvolvedores.

Existem dois diretórios importantes que devemos saber a composição. O diretório Assets e o diretório Adm. Ambos estão localizadas no diretório principal do projeto Unity do aplicativo.

No primeiro se coloca tudo que entra no desenvolvimento do jogo. Entre as coisas que estão lá dentro temos os scripts, músicas, prefabs, etc... Toda vez que se quer modificar algo no jogo, um dos arquivos dentro desta pasta será modificado.

No diretório adm são colocados arquivos que não são usados no desenvolvimento do jogo, mas ainda sim são importantes. Entre as coisas presentes lá temos o relatório que documenta a criação do aplicativo (como este que estão lendo), a política de privacidade que usamos no lançamento do aplicativo, a chave de publicação, e outros detalhes.

Dentro do diretório Assets existe uma estrutura interna de pastas que dispõe recursos utilizados no desenvolvimento de forma organizada e escalável. Existem recursos gerais e recursos específicos de fases, eles são dispostos da seguinte forma:

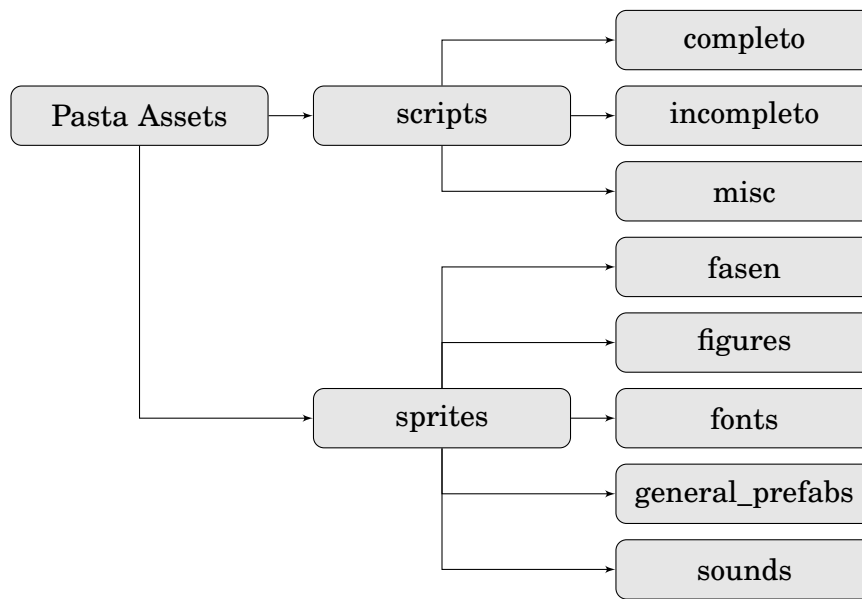


Fig. 2.1: Sistema de arquivos da pasta Assets. O que fora ignorado não tem importância para o desenvolvimento atual.

Dentro da pasta scripts temos três diretórios. Um para programas em c# completos, ou seja, já funcionais para o que se propõem. Outro para programas incompletos (coisas que estão em desenvolvimento devem ser postas aqui, até segunda ordem). E uma terceira pasta para scripts em geral, ou seja, para testes de alguma funcionalidade.

Dentro do diretório sprites encontram-se todo tipo de asset utilizado no jogo. Nas pastas fonts, general_prefabs, sounds e figures estão assets de uso geral no jogo, enquanto nas pastas fasen estão assets específicos de alguma fase.

- fasen: Assets específicos de uso na fase de número n. Aqui encontra-se os prefabs dos obstáculos e os componentes do background dinâmico.
- figures: estão assets visuais de uso geral, como as logos das universidades, os assets do background dinâmico inicial, a skin do player e elementos visuais da UI.
- fonts: encontram-se todas as fonts utilizadas no aplicativo.
- general_prefabs: encontram-se prefabs relacionados a UI e aos componentes puramente lógicos do jogo, cada qual em sua pasta dedicada (UI_prefabs e Game_rules).
- sounds: Nesta pasta encontram-se as músicas e sons do jogo.

Diretamente dentro da pasta Assets também encontra-se o arquivo "Player.Scene" que é a única cena do jogo. Nela encontram-se todas as referências a todos os elementos exibidos dentro do jogo.

2.2 Disposição de Game Objects

A plataforma Unity permite a divisão de objetos lógicos do jogo em "Game Objects", estas entidades são unidades de informação. Nelas é possível se armazenar qualquer tipo de elemento que possa aparecer

dentro do jogo, inclusive outros "Game Objects". Tal procedimento permite dividir os elementos lógicos da cena em unidades funcionais. Essas unidades podem então ser salvas de forma não volátil nos chamados "prefabs".

Dessa forma, dividir em unidades funcionais os elementos lógicos do jogo e separá-los fisicamente na memória do projeto resulta em muitos benefícios. Um deles é a organização e a escalabilidade. Torna-se muito fácil referir-se a um elemento do jogo quando a estrutura lógica é bem feita. Basta dar as coordenadas do elemento a partir dos Game Objects que a englobam. A estrutura ramificada garante um ambiente organizado mesmo com centenas de elementos na tela.

Outra vantagem está no controle de versão. Quando esses elementos lógicos são divididos por funcionalidade de forma física na memória, isso significa que ao se alterar elementos referentes a uma unidade, essas alterações estarão confinadas aquela unidade, não interferindo em outras. Tal propriedade permite que muitos integrantes da equipe de desenvolvimento trabalhem simultaneamente sem que editem os mesmos arquivos ao trabalharem em áreas diferentes do aplicativo. Isso facilita o processo de unificar o trabalho dos integrantes da equipe e garante menos problemas na utilização de sistemas de controle de versão como o "Git", ferramenta utilizada pelo grupo.

A seguir encontra-se a estrutura destes Game Objects na cena principal de jogo:

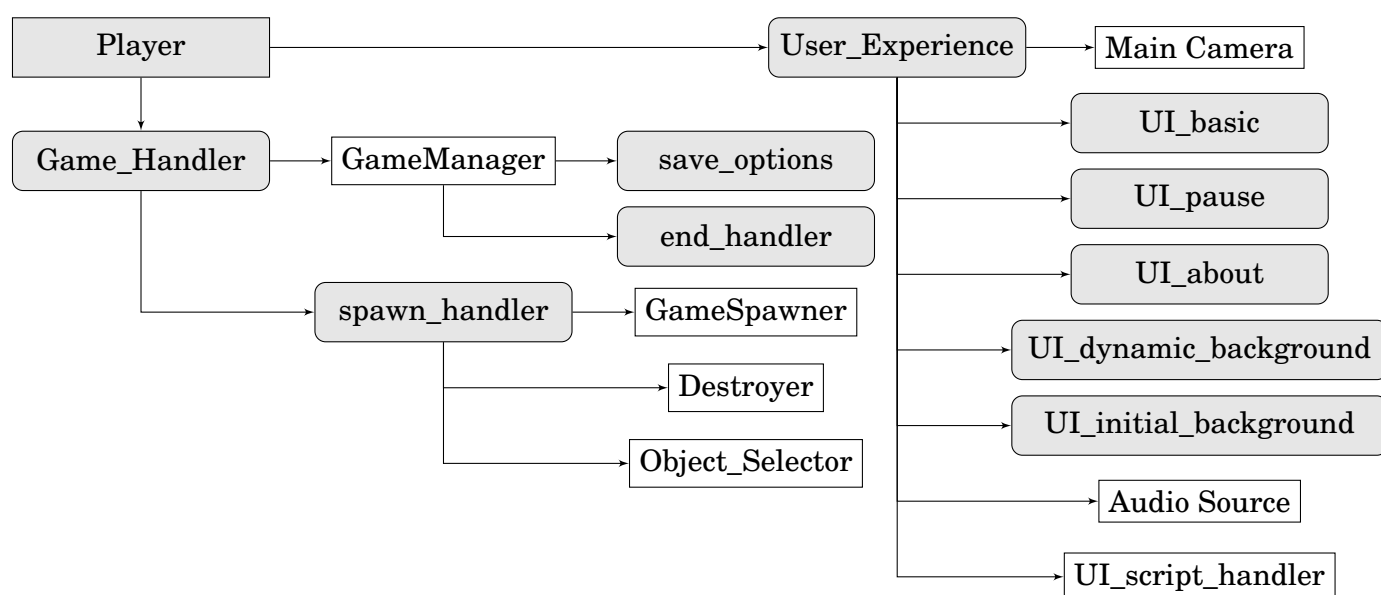


Fig. 2.2: Árvore de objetos de jogo, prefabs em azul.

Além dos elementos mostrados acima, temos também o objeto "player" e o objeto "EventSystem", ambos dentro somente da cena. O desenvolvedor que precisar alterar estes elementos é o único que terá a liberdade de fazer alterações no arquivo "Player.scene". Qualquer outra alteração será feita nos dois prefabs principais ou nos outros que estejam dentro destes.

Na Figura 2.2 podemos ver que existem Game Objects em azul e em branco. Os game objects em azul são os prefabs, ou seja, as alterações feitas neles não são gravadas nos objetos de jogo que os contém. Já os elementos em branco não são. Dessa forma, qualquer alteração feita neles serão salvas no prefab que o contém.

Por exemplo, se o desenvolvedor fizer uma alteração no objeto "save_options", tal alteração não aparecerá no prefab "Game_Handler", mas se a alteração for feita em "Game Manager", aparecerá. Também, se um desenvolvedor fizer alterações em "Audio Source", isso será salvo em "User_Experience". Já se a

alteração for feita em "UI_basic", não aparecerá, mas sim no próprio prefab "UI_basic".

A única forma de se causar alterações no arquivo da cena é se forem feitas alterações nos objetos "player" e "EventSystem". Eles estão localizados junto com "User_Experience" e "Game_Handler".

2.3 Ecossistema lógico

O aplicativo, como demonstrado anteriormente, já possui uma variedade de game objects e assets implementados. Além deles também já foram desenvolvidos muitos scripts na linguagem C#. Esses scripts estão presentes nos vários game objects mostrados anteriormente (Fig.2.2). Também vale a pena lembrar que estão dentro da pasta scripts (Fig.2.1).

Dentre os scripts mais simples, temos:

- Destroy_obstacles.cs (Fig.2.3) (spawn_handler > Destroyer): Tem instruções simples de destruir qualquer objeto que colidir com elemento inquilino.
- Object_Selector.cs (Fig.2.7) (spawn_handler > Object_Selector): Retorna um prefab de obstáculo conforme fase atual.
- Game_Manager.cs (Fig.2.8) (Player > Game_Handler): Fornece informações gerais e contabiliza o estado do jogo (fase e estado).
- audio_controls.cs (Fig.2.9) (User_Experience > Audio Source): Habilita o controle de volume presente na interface do menu pause.
- move.cs (Fig.2.4) (none): Implementa a movimentação dos obstáculos.
- movimentação.cs (Fig.2.6) (Player > player): Implementa a movimentação do jogador.
- interface_handler (Fig.2.5) (User_Experience > UI_script_handler): Possui as funções de todos os botões da interface.

```
1      using UnityEngine;
2
3      public class Destroy_obstacles : MonoBehaviour
4      {
5
6          // Destroi tudo que encostar neste objeto
7          void OnCollisionEnter2D(Collision2D coll)
8          {
9              Destroy(coll.gameObject);
10         }
11     }
```

Fig. 2.3: Código do script "Destroy_obstacles.cs".

```
1
2      using UnityEngine;
3
4
5      public class move : MonoBehaviour
6      {
7          public float speed;
8          void Start()
9          {
10             this.GetComponent<Rigidbody2D>().velocity = new Vector2(0, -
                speed);
11          }
12          public float Get_velocity(){return speed;}
13
14      }
```

Fig. 2.4: Código do script "move.cs".

```

1
2     using UnityEngine;
3     using UnityEngine.UI;
4
5     public class interface_handler : MonoBehaviour
6     {
7         // Game references:
8         public GameObject Basic_UI;
9         public GameObject Pause_UI;
10        public GameObject About_UI;
11        public int score = 0;
12        public Text scoreText;
13
14        void Start() {
15            // Turn on ui basic
16            Basic_UI.SetActive(true);
17        }
18        //! Função chamada uma vez por frame. CUIDADO com o que se coloca
19        // aqui.
20        void Update() {
21            if(score < 1000){
22                scoreText.text = score + " m ";
23            }else{
24                scoreText.text = score/10 + " km ";
25            }
26            // Contador simples para teste. No futuro referenciaremos
27            // script "phase_handler"
28            if(Input.GetKeyDown(KeyCode.Space)){
29                score = score + 100;
30            }
31        }
32        public void Pause(){Basic_UI.SetActive(false);Pause_UI.SetActive(
33            true);Time.timeScale = 0f;}
34        public void exit(){Application.Quit();}
35        public void Resume(){Basic_UI.SetActive(true);Pause_UI.SetActive(
36            false);Time.timeScale = 1f;}
37        public void About(){About_UI.SetActive(true);Pause_UI.SetActive(
38            false);}
39        public void exit_about(){About_UI.SetActive(false);Pause_UI.
40            SetActive(true);}
41        public void instagram_about(){Application.OpenURL("https://www.
42            instagram.com/equipe_epta/");}
43        public void facebook_about(){Application.OpenURL("https://www.
44            facebook.com/eptarocketdesign/");}
45        public void femec_about(){Application.OpenURL("http://www.mecanica.
46            ufu.br/");}
47        public void felt_about(){Application.OpenURL("http://www.felt.ufu
48            .br/");}
49        public void facic_about(){Application.OpenURL("http://www.facic.ufu
50            .br/");}
51        public void faced_about(){Application.OpenURL("http://www.faced.ufu
52            .br/");}
53        public void fagen_about(){Application.OpenURL("http://www.fagen.ufu
54            .br/");}
55        public void ufu_about(){Application.OpenURL("http://www.ufu.br/");}
56    }

```

Fig. 2.5: Código do script "interface_handler.cs".

```
1
2      using UnityEngine;
3
4      public class movimentação : MonoBehaviour
5      {
6          public float speed;
7          private Rigidbody2D rb;
8          private Vector3 localScreenWidth;
9
10         void Start()
11         {
12             rb = GetComponent<Rigidbody2D>();
13         }
14
15         void FixedUpdate()
16         {
17
18             // Button clicking detection
19             if(Input.GetMouseButton(0))
20             {
21                 Vector3 touchPos = Camera.main.ScreenToWorldPoint(Input.
22                     mousePosition);
23                 if(touchPos.x < 0){
24                     rb.velocity = new Vector2( -speed, rb.velocity.y);
25                 }
26                 else{
27                     rb.velocity = new Vector2( speed, rb.velocity.y);
28                 }
29             }
30
31             localScreenWidth = Camera.main.ScreenToWorldPoint(new Vector3(
32                 Screen.width, Screen.height, 0));
33
34             if (transform.position.x >= 1.09 * localScreenWidth.x)
35             {
36                 transform.position = new Vector3(- 1.07f * localScreenWidth
37                     .x, transform.position.y, transform.position.z);
38             }
39             else if (transform.position.x <= -1.09 * localScreenWidth.x)
40             {
41                 transform.position = new Vector3( 1.07f * localScreenWidth.
42                     x, transform.position.y, transform.position.z);
43             }
44         }
45     }
46 }
```

Fig. 2.6: Código do script "movimentação.cs".

```

1      using System.Collections.Generic;
2      using UnityEngine;
3      public class Object_selector : MonoBehaviour
4      {
5          public GameObject alien;
6          public GameObject balao_1;
7          ... (obstáculos)
8          public GameObject satelite_2;
9          private GameObject Game_manager;
10         private static GameObject obstacle;
11         private GameObject[][] actual_stage;
12         private int stage_index;
13         private int obstacle_index;
14         private List<int> stage_length_list;
15         private int stage_length;
16
17         void Start()
18         {
19             // Acha o script game manager via tag
20             Game_manager = GameObject.FindWithTag("Game_manager");
21
22         }
23
24         public GameObject Get_Obstacle(){
25
26             Debug.Log(Game_manager.GetComponent<Game_Manager>().Get_phase()
27                 );
28             stage_index = 0;
29             actual_stage = new GameObject[3][];
30             actual_stage[0] = new GameObject[4];
31             actual_stage[0][0] = nuvem_1_0;
32             actual_stage[0][1] = nuvem_1_1;
33             actual_stage[0][2] = nuvem_3_0;
34             actual_stage[0][3] = balao_1;
35             actual_stage[1] = new GameObject[3];
36             actual_stage[1][0] = meteoro_1_0;
37             actual_stage[1][1] = meteoro_2_0;
38             actual_stage[1][2] = meteoro_3_0;
39             actual_stage[2] = new GameObject[4];
40             actual_stage[2][0] = satelite_1;
41             actual_stage[2][1] = satelite_2;
42             actual_stage[2][2] = satelite_3;
43             actual_stage[2][3] = alien;
44             stage_length_list.Add(4);
45             stage_length_list.Add(3);
46             stage_length_list.Add(4);
47
48             stage_length = stage_length_list[stage_index];
49             obstacle_index = (int)Random.Range(0, stage_length);
50             obstacle = actual_stage[stage_index][obstacle_index];
51             return obstacle;
52         }
53     }

```

Fig. 2.7: Código do script "Object_Selector.cs".

```

1
2     using UnityEngine;
3
4     public class Game_Manager : MonoBehaviour{
5
6         // Refere-se à fase
7         private int phase = 0;
8         // Refere-se ao jogador
9         private GameObject player;
10        // Refere-se ao tempo relativo da fase
11        private float phase_time;
12        // Tempo da próxima fase
13        private float next_phase;
14        // Refere-se ao tempo total de jogo
15        private float game_time;
16        // Define a duração de cad fase
17        private float[] phase_plan = new float[2]{10.0f, 10.0f};
18        void Start()
19        {
20            // Procura pelo objeto de jogo do jogador
21            player = GameObject.FindWithTag("Player");
22            // Tempo inicial, após dado play
23            game_time = Time.time;
24            next_phase = 0.0f;
25        }
26        void Update()
27        {
28            // Phase Query from time
29            if(Time.time > next_phase){
30                if (phase >= phase_plan.Length){
31                    // final com fase infinita
32                    next_phase = 36000000.0f;
33                }else{
34                    // Registra tempo de conversão para nova fase
35                    next_phase = phase_plan[phase] + Time.time;
36                }
37                // Passa a fase e registra tempo
38                phase++;
39                phase_time = Time.time;
40            }
41        }
42        // Pega posição em x do jogador -2.6 < x < 2.6
43        public float Get_player_x(){return player.GetComponent<Transform>().
        .position.x;}
44        // Pega objeto do jogador
45        public GameObject Get_player(){return player;}
46        // Pega fase atual (começa em 1)
47        public int Get_phase(){return phase;}
48        // Pega tempo de fase atual
49        public float Get_phase_time(){return Time.time - phase_time;}
50        // Pega fração de completude de fase de 0 a 1
51        public float Get_phase_fraction(){return (Time.time - phase_time)/
        phase_plan[phase];}
52        // Pegar tempo global de execução
53        public float Get_time(){return Time.time - game_time;}
54    }

```

Fig. 2.8: Código do script "Game_Manager.cs".


```
1      using UnityEngine;
2
3      public class audio_controls: MonoBehaviour
4      {
5          // Reference to the scene audio controll
6          private GameObject music_volume_control;
7
8          // makes the starting volume change in UI
9          [SerializeField]
10         private float volume;
11
12         // Start is called before the first frame update
13         void Start()
14         {
15             // Acha o game audio principal, com a música do jogo via tag (
16             //   GameObject Audio Source)
17             music_volume_control = GameObject.FindWithTag("audio_control");
18
19             // Declara o volume
20             music_volume_control.GetComponent<AudioSource>().volume =
21                 volume;
22
23         }
24
25         // Quando o valor do volume muda, atualiza o valor no Audio source
26         // da música
27         public void volume_controll(float volume){
28             music_volume_control.GetComponent<AudioSource>().volume =
29                 volume;
30         }
31     }
```

Fig. 2.9: Código do script "audio_controls.cs".

Capítulo 3

Plano de gestão para 2021

Tabela 3.1: Planejamento escalonado

Recurso	Semana 1 (01/16)	Semana 2 (01/23)	Semana 3 (01/30)	Semana 4 (02/06)	Semana 5 (02/13)	Semana 6 (02/20)	Semana 7 (02/27)
Background dinâmico Memória não volátil Tela inicial dinâmica Recursos de fim de jogo Mecânica de Spawn Propaganda funcional							

Capítulo 4

Planos futuros