

# Porte do jogo *Traveling Will* para o *Nintendo Game Boy Advance*

Igor Ribeiro Barbosa Duarte e Vítor Barbosa de Araujo

Faculdade Gama  
Universidade de Brasília  
Brasil

10 de dezembro de 2018

# Objetivo geral

- O objetivo geral deste trabalho é reescrever o jogo *Traveling Will*, desenvolvido originalmente para PC na disciplina de Introdução aos Jogos Eletrônicos, para o *Nintendo Game Boy Advance*

# Objetivos específicos

Os objetivos específicos são:

- Comprimir imagens e músicas do jogo original para reduzir o uso de memória;
- Criar módulos para renderização de imagens e texto;
- Criar módulos para manipulação de *inputs* dos botões e carregamento de áudio;
- Criar módulos para detecção de colisões e manipulação de eventos;
- Criar o jogo;
- Executar e testar o jogo desenvolvido na plataforma escolhida.

# Objetivos específicos

Os objetivos específicos são:

- Comprimir imagens e músicas do jogo original para reduzir o uso de memória;
- Criar módulos para renderização de imagens e texto;
- Criar módulos para manipulação de *inputs* dos botões e carregamento de áudio;
- Criar módulos para detecção de colisões e manipulação de eventos;
- Criar métodos para carregamento do *level design* das fases do jogo;
- Executar e testar o jogo desenvolvido na plataforma escolhida.

# Objetivos específicos

Os objetivos específicos são:

- Comprimir imagens e músicas do jogo original para reduzir o uso de memória;
- Criar módulos para renderização de imagens e texto;
- Criar módulos para manipulação de *inputs* dos botões e carregamento de áudio;
- Criar módulos para detecção de colisões e manipulação de eventos;
- Criar métodos para carregamento do *level design* das fases do jogo;
- Executar e testar o jogo desenvolvido na plataforma escolhida.

# Objetivos específicos

Os objetivos específicos são:

- Comprimir imagens e músicas do jogo original para reduzir o uso de memória;
- Criar módulos para renderização de imagens e texto;
- Criar módulos para manipulação de *inputs* dos botões e carregamento de áudio;
- Criar módulos para detecção de colisões e manipulação de eventos;
- Criar métodos para carregamento do *level design* das fases do jogo;
- Executar e testar o jogo desenvolvido na plataforma escolhida.

# Objetivos específicos

Os objetivos específicos são:

- Comprimir imagens e músicas do jogo original para reduzir o uso de memória;
- Criar módulos para renderização de imagens e texto;
- Criar módulos para manipulação de *inputs* dos botões e carregamento de áudio;
- Criar módulos para detecção de colisões e manipulação de eventos;
- Criar métodos para carregamento do *level design* das fases do jogo;
- Executar e testar o jogo desenvolvido na plataforma escolhida.

# Objetivos específicos

Os objetivos específicos são:

- Comprimir imagens e músicas do jogo original para reduzir o uso de memória;
- Criar módulos para renderização de imagens e texto;
- Criar módulos para manipulação de *inputs* dos botões e carregamento de áudio;
- Criar módulos para detecção de colisões e manipulação de eventos;
- Criar métodos para carregamento do *level design* das fases do jogo;
- Executar e testar o jogo desenvolvido na plataforma escolhida.



# Ferramentas

Para o porte do jogo foram utilizadas as seguintes ferramentas:

- C++
- *devkitARM*
- *GIMP*
- *GRIT*
- *svconv*, *libressl* e *modplug tracker*
- *SDL*
- *SDL\_mixer*
- *SDL\_image*

# Ferramentas

Para o porte do jogo foram utilizadas as seguintes ferramentas:

- *C++*
- *devkitARM*
- *GIMP*
- *GRIT*
- *avconv*, *librosa* e *modplug tracker*
- *SDL*
- *SDL\_mixer*
- *SDL\_image*
- *SDL\_ttf*

# Ferramentas

Para o porte do jogo foram utilizadas as seguintes ferramentas:

- *C++*
- *devkitARM*
- *GIMP*
- *GRIT*
- *avconv, librosa e modplug tracker*
- *VisualBoyAdvance-M*
- *SDL\_mixer*
- *LibRetro*

# Ferramentas

Para o porte do jogo foram utilizadas as seguintes ferramentas:

- *C++*
- *devkitARM*
- *GIMP*
- *GRIT*
- *avconv*, *librosa* e *modplug tracker*
- *VisualBoyAdvance-M*
- *Nintendo DS*
- *Libretro*

# Ferramentas

Para o porte do jogo foram utilizadas as seguintes ferramentas:

- *C++*
- *devkitARM*
- *GIMP*
- *GRIT*
- *avconv*, *librosa* e *modplug tracker*
- *VisualBoyAdvance-M*
- *Nintendo DS*
- *EZFlash II*

# Ferramentas

Para o porte do jogo foram utilizadas as seguintes ferramentas:

- *C++*
- *devkitARM*
- *GIMP*
- *GRIT*
- *avconv*, *librosa* e *modplug tracker*
- *VisualBoyAdvance-M*
- *Nintendo DS*
- *EZFlash II*

# Ferramentas

Para o porte do jogo foram utilizadas as seguintes ferramentas:

- *C++*
- *devkitARM*
- *GIMP*
- *GRIT*
- *avconv*, *librosa* e *modplug tracker*
- *VisualBoyAdvance-M*
- *Nintendo DS*
- *EZFlash II*

# Ferramentas

Para o porte do jogo foram utilizadas as seguintes ferramentas:

- *C++*
- *devkitARM*
- *GIMP*
- *GRIT*
- *avconv*, *librosa* e *modplug tracker*
- *VisualBoyAdvance-M*
- *Nintendo DS*
- *EZFlash II*



# Desenvolvimento da *engine*

Para este trabalho foi desenvolvida a *gbengine*.

- Módulos implementados: *input*, vídeo, gerenciador de memória, áudio e física;
- **Módulo de input:** detectar pressionamento dos botões do GBA.
- **Módulo de vídeo:** renderizar, animar e atualizar imagens;
- **Módulo gerenciador de memória:** garantir alocação segura e eficiente de memória;
- **Módulo de áudio:** carregar e reproduzir arquivos de áudio;
- **Módulo de física:** simular a ação de gravidade e detectar colisões entre objetos do jogo.

# Desenvolvimento da *engine*

Para este trabalho foi desenvolvida a *gbengine*.

- Módulos implementados: *input*, vídeo, gerenciador de memória, áudio e física;
- **Módulo de input:** detectar pressionamento dos botões do GBA.
- **Módulo de vídeo:** renderizar, animar e atualizar imagens;
- **Módulo gerenciador de memória:** garantir alocação segura e eficiente de memória;
- **Módulo de áudio:** iniciar e parar músicas de fundo;
- **Módulo de física:** gerenciar a colisão de objetos e detectar colisões entre objetos do jogo.

# Desenvolvimento da *engine*

Para este trabalho foi desenvolvida a *gbengine*.

- Módulos implementados: *input*, vídeo, gerenciador de memória, áudio e física;
- **Módulo de input:** detectar pressionamento dos botões do GBA.
- **Módulo de vídeo:** renderizar, animar e atualizar imagens;
- **Módulo gerenciador de memória:** garantir alocação segura e eficiente de memória;
- **Módulo de áudio:** iniciar e parar músicas de fundo;
- **Módulo de física:** simular a ação de gravidade e detectar colisões entre objetos do jogo;

# Desenvolvimento da *engine*

Para este trabalho foi desenvolvida a *gbengine*.

- Módulos implementados: *input*, vídeo, gerenciador de memória, áudio e física;
- **Módulo de input:** detectar pressionamento dos botões do GBA.
- **Módulo de vídeo:** renderizar, animar e atualizar imagens;
- **Módulo gerenciador de memória:** garantir alocação segura e eficiente de memória;
- **Módulo de áudio:** iniciar e parar músicas de fundo;
- **Módulo de física:** simular a ação de gravidade e detectar colisões entre objetos do jogo;

# Desenvolvimento da *engine*

Para este trabalho foi desenvolvida a *gbengine*.

- Módulos implementados: *input*, vídeo, gerenciador de memória, áudio e física;
- **Módulo de input:** detectar pressionamento dos botões do GBA.
- **Módulo de vídeo:** renderizar, animar e atualizar imagens;
- **Módulo gerenciador de memória:** garantir alocação segura e eficiente de memória;
- **Módulo de áudio:** iniciar e parar músicas de fundo;
- **Módulo de física:** simular a ação de gravidade e detectar colisões entre objetos do jogo;

# Desenvolvimento da *engine*

Para este trabalho foi desenvolvida a *gbengine*.

- Módulos implementados: *input*, vídeo, gerenciador de memória, áudio e física;
- **Módulo de input:** detectar pressionamento dos botões do GBA.
- **Módulo de vídeo:** renderizar, animar e atualizar imagens;
- **Módulo gerenciador de memória:** garantir alocação segura e eficiente de memória;
- **Módulo de áudio:** iniciar e parar músicas de fundo;
- **Módulo de física:** simular a ação de gravidade e detectar colisões entre objetos do jogo;

# Módulo de input

- Detectar pressionamento dos botões
- *Bitmask* onde o valor 0 indica pressionamento

# Módulo de input

```
5 void check_buttons_states() {
6     update_counter++;
7
8     for(int i = 0; i < N_BUTTON; i++) {
9         int pressed = !((*buttons_mem) & (1 << i));
10
11         if (pressed == pressed_state[i]) continue;
12
13         pressed_state[i] = pressed;
14         key_update[i] = update_counter;
15     }
16 }
17
18 bool pressed(int button) {
19     return pressed_state[32 - __builtin_clz(button) - 1] && key_update[32
        - __builtin_clz(button) - 1] == update_counter;
20 }
21
22 bool pressing(int button) {
23     return pressed_state[32 - __builtin_clz(button) - 1];
24 }
```

Figura: Código da classe *input*



# Módulo de input



Figura: *Demo do input*

# Módulo de input

```
9  int main() {
10      reset_dispcnt();
11      set_video_mode(3);
12      enable_background(2);
13
14      while(1) {
15          check_buttons_states();
16
17          for(int i=0;i<=9;i++){
18              int padding = i + 10;
19
20              if (pressing(1 << i)) {
21                  for(int x=-2;x<=2;x++) {
22                      for(int y=-2;y<=2;y++) {
23                          vid_mem[(50 + x) * 240 + (20 + y + i * 10)] = RED;
24                      }
25                  }
26              }
27              else {
28                  for(int x=-2;x<=2;x++) {
29                      for(int y=-2;y<=2;y++) {
30                          vid_mem[(50 + x) * 240 + (20 + y + i * 10)] = 0;
31                      }
32                  }
33              }
34          }
35      }
```

Figura: Código do *demo* de *input*

# Módulo de vídeo

- Classe Texture: renderização e animação de *sprites*
- Construtor por cópia
- Classe Background: renderização e *scroll* de *backgrounds*
- Escolha da paleta do *background* no momento da conversão

# Módulo de vídeo

- Classe Texture: renderização e animação de *sprites*
- Construtor por cópia
- Classe Background: renderização e *scroll* de *backgrounds*
- Escolha da paleta do *background* no momento da conversão

# Módulo de vídeo

- Classe Texture: renderização e animação de *sprites*
- Construtor por cópia
- Classe Background: renderização e *scroll* de *backgrounds*
- Escolha da paleta do *background* no momento da conversão

# Módulo de vídeo

- Classe Texture: renderização e animação de *sprites*
- Construtor por cópia
- Classe Background: renderização e *scroll* de *backgrounds*
- Escolha da paleta do *background* no momento da conversão

# Módulo de vídeo

```
1 Texture::Texture(uint32_t num_sprites, const uint16_t *pallette, uint32_t
    pallette_len,
2     const unsigned int *tiles, uint32_t tiles_len, enum bits_per_pixel bpp
    = _8BPP)
3 {
4     this->pallette = pallette;
5     this->pallette_len = pallette_len;
6     this->pallette_id = 0;
7     this->bpp = bpp;
8     this->num_sprites = num_sprites;
9     this->num_tiles = tiles_len / ((bpp == _4BPP) ? 32 : 64);
10    this->tiles_per_sprite = num_tiles / num_sprites;
11    this->tiles = tiles;
12    this->tiles_len = tiles_len;
13
14    memory_manager = MemoryManager::get_memory_manager();
15
16    set_sprite_pal();
17    set_sprite();
18    oam_entry = memory_manager->alloc_oam_entry();
19
20    metadata.tid = tile_base * ((bpp == _4BPP) ? 1 : 2);
21    metadata.pb = pallette_id;
22 }
```

Figura: Construtor da classe Texture

# Módulo gerenciador de memória

- Garantir alocação segura e eficiente de memória
- Gerenciamento com partições variáveis



# Módulo gerenciador de memória

Funcionamento do gerenciador de memória:

- Inicializar os ponteiros para as regiões de memória
- Quando há chamada de alocação, procura pelo primeiro espaço disponível
- Verificar espaço disponível
- Liga posição no bitset
- Retorna o ponteiro encontrado

# Módulo gerenciador de memória

```
1 struct attr {  
2     // attr0  
3     uint8_t y;  
4     uint8_t om : 2;  
5     uint8_t gm : 2;  
6     uint8_t mos : 1;  
7     uint8_t cm : 1;  
8     uint8_t sh : 2;  
9     // attr1  
10    uint16_t x : 9;  
11    uint8_t aid : 5;  
12    uint8_t sz : 2;  
13    // attr2  
14    uint16_t tid : 10;  
15    uint8_t pr : 2;  
16    uint8_t pb : 4;  
17    // attr3  
18    uint16_t filler;  
19 };
```

Figura: Uso de *bitfields* nos metadados das *sprites*

# Módulo gerenciador de memória

```
1 class MemoryManager {  
2     private:  
3         MemoryManager *instance;  
4     public:  
5         MemoryManager *get_memory_manager() {  
6             if (!instance) {  
7                 instance = new MemoryManager();  
8             }  
9             return instance;  
10        }  
11 };
```

Figura: Uso de *singleton* na classe MemoryManager

# Módulo de física

- Checar se os objetos estão colidindo
- Chamar `on_collision()` do alvo

# Módulo utilitário

Código *assembly* com chamada para `vbaprint()`

```
1 .arm
2 .global vbaprint
3 .type vbaprint STT_FUNC
4 .text
5 vbaprint:
6     swi 0xFF0000
7     bx lr
```

Figura: Chamada para `vbaprint()`

# Módulo utilitário

## Função print()

```
1 int print(const char *fmt, ...) {  
2     va_list args;  
3     va_start(args, fmt);  
4  
5     char buffer[4096];  
6  
7     int rc = vsnprintf(buffer, sizeof(buffer), fmt, args);  
8  
9     vbaprint(buffer);  
10    va_end(args);  
11  
12    return rc;  
13 }
```

Figura: Função print()

# Módulo utilitário

## Função mem16cpy()

```
1 void mem16cpy(volatile void *dest, const void *src, size_t n)
2 {
3     if (n & 1) {
4         print("Size must be even");
5     }
6
7     for (int i = 0; i < n / 2; i++) {
8         *(((volatile uint16_t *)dest) + i) = *(((uint16_t *)src) + i);
9     }
10 }
```

Figura: Função mem16cpy()

# Módulo utilitário

## Função vsync()

```
1 void vsync() {  
2     while(REG_VCOUNT >= 160);  
3     while(REG_VCOUNT < 160);  
4 }
```

Figura: Função vsync()



# Adaptação das músicas do jogo

- *Parser* do level design original
- Reprodução de ondas quadradas
- Reduzir a frequência das músicas originais

# Adaptação das músicas do jogo

- *Parser* do level design original
- Reprodução de ondas quadradas
- Reduzir a frequência das músicas originais

# Adaptação das músicas do jogo

- *Parser* do level design original
- Reprodução de ondas quadradas
- Reduzir a frequência das músicas originais

# Adaptação das músicas do jogo

```
1 // header file
2 #ifndef FASE1_H
3 #define FASE1_H
4
5 #define fase1_notes_len 70
6 #define fase1_tempos_len 70
7
8 extern const int fase1_notes[70];
9 extern const int fase1_tempos[70];
10
11 #endif
12
13 // source code
14 #include "fase1.h"
15
16 const int fase1_notes[70] = {
17     1, 0, 1, 0, 10, 8, 5, 3, 2, 3, 2, 0, 10, 7, 7, 6, 7, 6, 3, 2, 0, 10,
18     9, 7, 5, 7, 9, 10, 9, 10, 0, 2, 1, 0, 1, 3, 1, 0, 10, 8, 5, 2, 3,
19     2, 3, 2, 3, 2, 3, 2, 10, 7, 7, 6, 7, 6, 3, 2, 0, 10, 9, 7, 5, 7, 9,
20     10, 9, 10, 0, 2
21 };
22
23 const int fase1_tempos[70] = {
24     4, 4, 4, 2, 2, 2, 2, 4, 4, 4, 2, 2, 2, 2, 4, 4, 4, 2, 2, 2, 2, 2,
25     2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 2, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1,
26     4, 4, 2, 1, 1, 2, 2, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
27     2, 2, 4
28 };
```

Figura: Arquivo de áudio gerado com o *parser*.

## Adaptação das músicas do jogo

```
15 void Sound::load_from_file(int notes_len, const int* notes, int
    tempos_len, const int* tempos) {
16     this->notes = notes;
17     this->notes_len = notes_len;
18     this->tempos = tempos;
19     this->tempos_len = tempos_len;
20 }
21
22 void Sound::play_note(int raw_note, int tempo) {
23     int note = raw_note % 12;
24     int octave = raw_note / 12;
25     REG_SND1FREQ = SFREQ_RESET | SND_RATE(note, octave);
26
27     for(int i=0; i<8*tempo; i++) {
28         vsync();
29     }
30 }
31
32 void Sound::play() {
33     for(int i=0; i < notes_len; i++) {
34         print("note %d tempo %d\n", notes[i], tempos[i]);
35         play_note(notes[i], tempos[i]);
36     }
37 }
```

Figura: Código para reprodução de ondas quadradas.

# Adaptação das músicas do jogo

```
void Sound::play_sample() {  
    mmFrame();  
  
    if (frame_count % frames_per_sample == 0) {  
        stop_sample();  
        start_sample(mod_list[sound_it]);  
  
        sound_it++;  
        sound_it %= mod_list.size();  
    }  
    frame_count++;  
}  
  
void Sound::load_music(const vector<int> &mod_list) {  
    this->mod_list = mod_list;  
    sound_it = 0;  
    frame_count = 0;  
}
```

Figura: Código para reprodução de arquivos mod.

# Adaptação das músicas do jogo

```
void Sound::start_sample(int mod_id) {  
    mmStart(mod_id, MM_PLAY_ONCE);  
}  
  
void Sound::stop_sample() {  
    mmStop();  
    mmPosition(1);  
    frame_count = 0;  
}
```

Figura: Código para reprodução de arquivos mod.

# Adaptação das imagens do jogo

- Proporção 3:1 em relação à altura das imagens
- Conversão das imagens utilizando a ferramenta *grit*



# Adaptação das imagens do jogo

- Proporção 3:1 em relação à altura das imagens
- Conversão das imagens utilizando a ferramenta `grit`

# Adaptação das imagens do jogo

```
1 $ grit nome-da-imagem.png -gB4 -ftc -Mw2 -Mh4
```

Figura: Comando para conversão das *sprites*

```
1 $ grit nome-da-imagem.png -gB4 -ftc -mRtf -mp0
```

Figura: Comando para conversão dos *backgrounds*

# Adaptação das imagens do jogo

- Tamanho do *background*
- *Scroll* horizontal dos *backgrounds* (registradores REGBGxHOFs)

# Adaptação das imagens do jogo

- Tamanho do *background*
- *Scroll* horizontal dos *backgrounds* (registradores REGBGxH0FS)

# Adaptação das imagens do jogo

```
1 void Background::update_self(uint64_t dt) {
2     if (dt % frames_to_skip == 0) {
3         m_x += m_speed_x;
4         m_y += m_speed_y;
5     }
6
7     switch(this->background_id) {
8         case 0:
9             REG_BG0H0FS = m_x;
10            REG_BG0V0FS = m_y;
11            break;
12        case 1:
13            REG_BG1H0FS = m_x;
14            REG_BG1V0FS = m_y;
15            break;
16        case 2:
17            REG_BG2H0FS = m_x;
18            REG_BG2V0FS = m_y;
19            break;
20        case 3:
21            REG_BG3H0FS = m_x;
22            REG_BG3V0FS = m_y;
23            break;
24        default:
25            print("Invalid background id\n");
26            break;
27    }
28 }
```

Figura: Scroll horizontal dos *backgrounds*

# Construção dos níveis do jogo

- *Parser para level design original*
- Reutilizar texturas das plataformas
- Carregar somente os itens visíveis

# Construção dos níveis do jogo

- *Parser* para *level design* original
- Reutilizar texturas das plataformas
- Carregar somente os itens visíveis

# Construção dos níveis do jogo

- *Parser* para *level design* original
- Reutilizar texturas das plataformas
- Carregar somente os itens visíveis



# Construção dos níveis do jogo

```
1 // neste exemplo, platform_idx corresponde ao indice da plataforma mais
  a direita da tela
2 while (1) {
3     if (platform_idx * platform_width <= m_backgrounds[0]->x() +
        screen_width) {
4         auto plat = q.front();
5         q.pop();
6
7         plat->set_x(platform_idx * platform_width - m_backgrounds[0]->x());
8         plat->set_y(platform_height[platform_idx]);
9
10        plat->collectable()->set_y(collectable_height[platform_idx]);
11        plat->collectable()->set_visibility(collectable_present[platform_idx
12        ]);
13        q.push(plat);
14    } else break;
15
16    platform_idx++;
17 }
```

Figura: Código para atualização das plataformas

# Transição entre os níveis do jogo

```
1  if (m_level->done()) {
2      if (m_level->next() == NEXT_LEVEL) {
3          current_level = (previous_playable_level + 1) % (LOADED_LEVELS + 1);
4          if (current_level == 0)
5              current_level++;
6      }
7      else {
8          previous_playable_level = current_level;
9          current_level = m_level->next();
10     }
11
12     delete m_level;
13
14     bool is_playable = current_level != LEVEL_MENU && current_level !=
        MENU_VICTORY
15         && current_level != MENU_DEFEAT;
16
17
18     m_level = new TWLevel(current_level, is_playable);
19 }
```

Figura: Transição entre níveis

# Classes do jogo

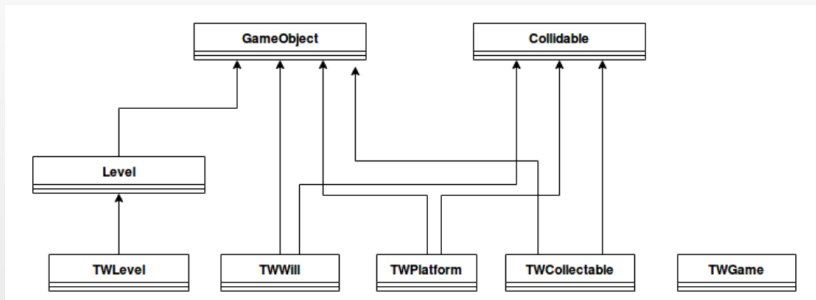


Figura: Diagrama de classes do jogo.

# Comparação entre o porte e o jogo original



Figura: Comparação entre os menus.

# Comparação entre o porte e o jogo original

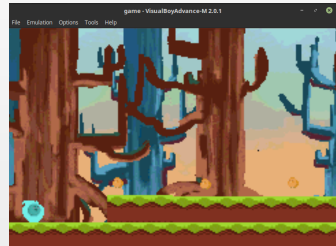


Figura: Comparação da primeira fase.

## Comparação entre o porte e o jogo original

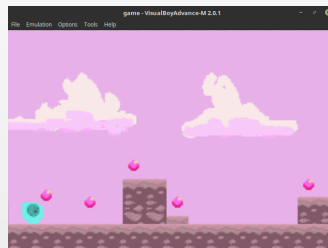
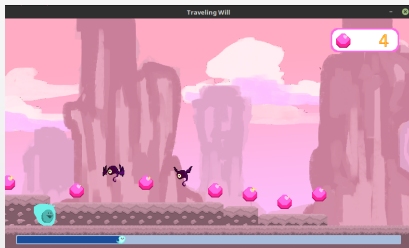


Figura: Comparação da segunda fase.

# Comparação entre o porte e o jogo original

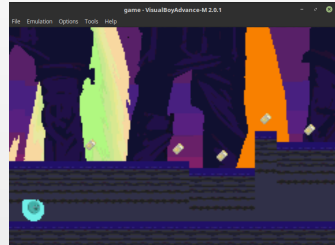
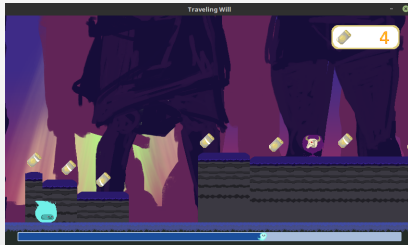


Figura: Comparação da terceira fase.

# Comparação entre o porte e o jogo original



Figura: Comparação da quarta fase.



# Comparação entre o porte e o jogo original

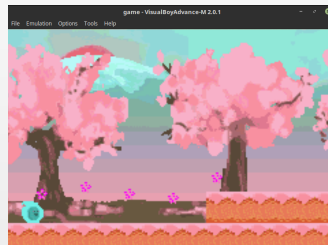
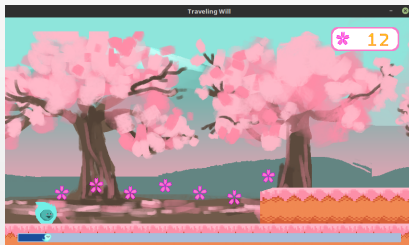


Figura: Comparação da quinta fase.

# Comparação entre o porte e o jogo original

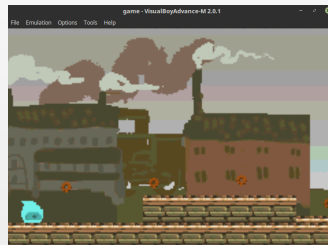
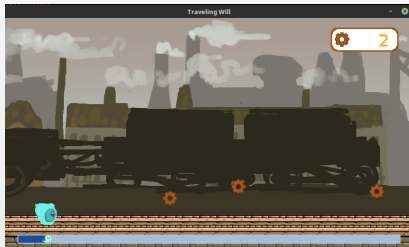


Figura: Comparação da sexta fase.

# Comparação entre o porte e o jogo original

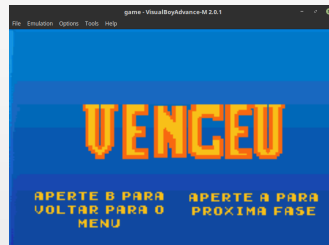
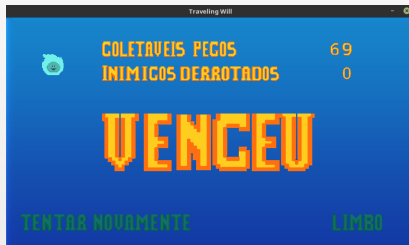


Figura: Comparação das telas de vitória.

# Comparação entre o porte e o jogo original

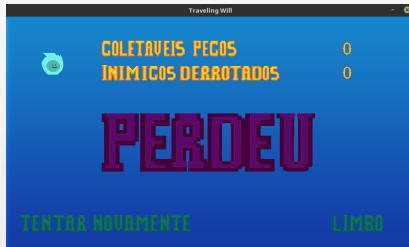


Figura: Comparação das telas de derrota.

# Considerações Finais

Principais impedimentos durante a execução do trabalho:

- Entender detalhes do *hardware do GBA*;
- Testar o jogo no *console*;
- Adaptação de imagens e músicas do jogo;

# Considerações Finais

Principais impedimentos durante a execução do trabalho:

- Entender detalhes do *hardware do GBA*;
- Testar o jogo no *console*;
- Adaptação de imagens e músicas do jogo;

# Considerações Finais

Principais impedimentos durante a execução do trabalho:

- Entender detalhes do *hardware do GBA*;
- Testar o jogo no *console*;
- Adaptação de imagens e músicas do jogo;

# Considerações Finais

Pontos de melhoria pós-execução do trabalho:

- Melhor priorização das tarefas a serem executadas;
- Testes mais frequentes no *console*



# Considerações Finais

Pontos de melhoria pós-execução do trabalho:

- Melhor priorização das tarefas a serem executadas;
- Testes mais frequentes no *console*

# Considerações Finais

*É possível portar o jogo *Traveling Will*, desenvolvido para PC pelos autores deste trabalho, para o Nintendo Gameboy Advance, no contexto de um trabalho de conclusão de curso, com performance e jogabilidade próximos da versão para computador?*

R: **Sim, é possível.**

# Trabalhos futuros

- Melhorar módulo de áudio para carregar efeitos sonoros e pausar músicas;
- Implementar carregamento e utilização de fontes;
- Adicionar elementos de HUD e seleção de fases;
- Salvar o estado do jogo em memória;

# Trabalhos futuros

- Melhorar módulo de áudio para carregar efeitos sonoros e pausar músicas;
- Implementar carregamento e utilização de fontes;
- Adicionar elementos de HUD e seleção de fases;
- Salvar o estado do jogo em memória;
- Implementar um desfragmentador de memória na classe `MemoryManager`

# Trabalhos futuros

- Melhorar módulo de áudio para carregar efeitos sonoros e pausar músicas;
- Implementar carregamento e utilização de fontes;
- Adicionar elementos de HUD e seleção de fases;
- Salvar o estado do jogo em memória;
- Implementar um desfragmentador de memória na classe `MemoryManager`

# Trabalhos futuros

- Melhorar módulo de áudio para carregar efeitos sonoros e pausar músicas;
- Implementar carregamento e utilização de fontes;
- Adicionar elementos de HUD e seleção de fases;
- Salvar o estado do jogo em memória;
- Implementar um desfragmentador de memória na classe `MemoryManager`

# Trabalhos futuros

- Melhorar módulo de áudio para carregar efeitos sonoros e pausar músicas;
- Implementar carregamento e utilização de fontes;
- Adicionar elementos de HUD e seleção de fases;
- Salvar o estado do jogo em memória;
- Implementar um desfragmentador de memória na classe `MemoryManager`

# Considerações Finais

Obrigado!