

# 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*
- *SDL2*
- *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

```
1  #include "input.h"
2
3  volatile unsigned int *buttons_mem = (volatile unsigned int *) 0
    x04000130;
4
5  void check_buttons_states() {
6      for(int i = 0; i < N_BUTTON; i++) {
7          pressed_state[i] = !((*buttons_mem) & (1 << i));
8      }
9  }
10
11 bool pressed(int button) {
12     return pressed_state[button];
13 }
```

Figura: Código da classe *input*



# Módulo de input

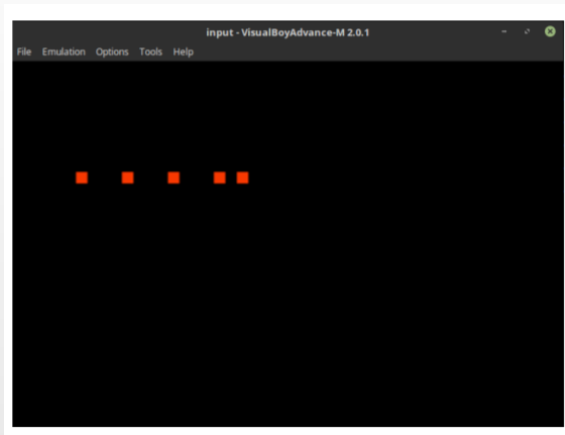


Figura: *Demo do input*

# 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

# 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

# Adaptação das imagens do jogo

# 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

# Comparação entre o porte e o jogo original



Figura: Comparação entre os menus.

# Comparação entre o porte e o jogo original

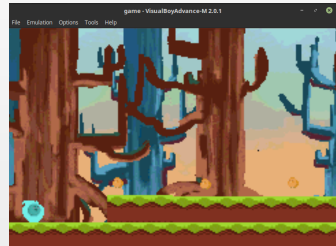


Figura: Comparação da primeira fase.

# Comparação entre o porte e o jogo original

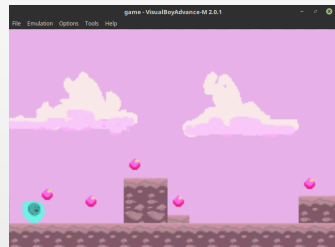
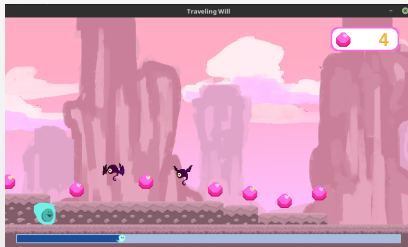


Figura: Comparação da segunda fase.

# Comparação entre o porte e o jogo original

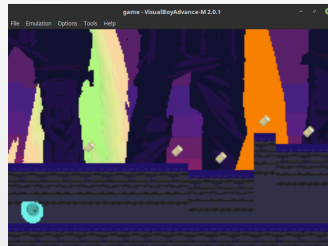
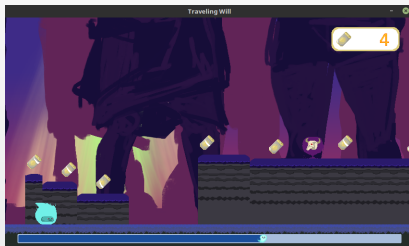


Figura: Comparação da terceira fase.

# Comparação entre o porte e o jogo original



Figura: Comparação da quarta fase.



# Comparação entre o porte e o jogo original

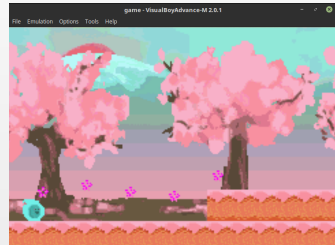
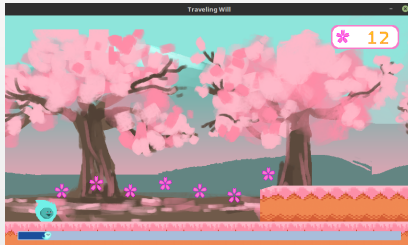


Figura: Comparação da quinta fase.

# Comparação entre o porte e o jogo original

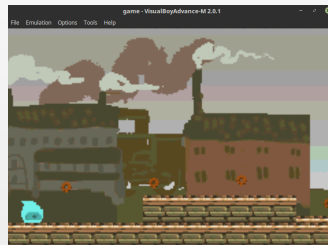


Figura: Comparação da sexta fase.

# Comparação entre o porte e o jogo original

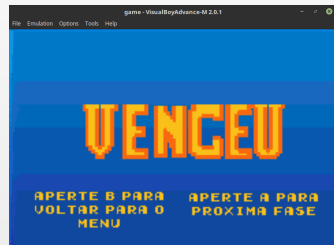
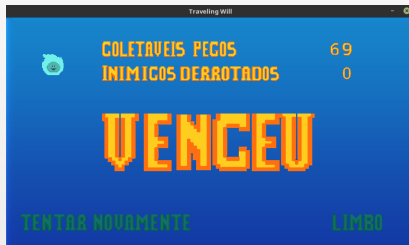


Figura: Comparação das telas de vitória.

# Comparação entre o porte e o jogo original

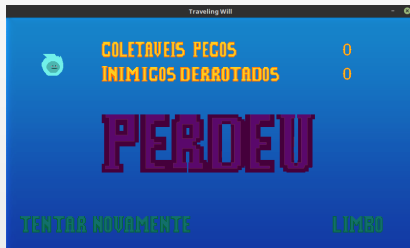


Figura: Comparação das telas de derrota.

# 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!