Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

PROFITABLE DUNGEONS

Gabriel de Oliveira Guedes Nogueira (10295496)
Mateus Zanetti Camargo Penteado (11219202)
Daniel Suzumura (11218921)

Documentação do jogo desenvolvido como trabalho da disciplina *SSC0140 - Sistemas Operacionais*, ministrada no 2o semestre de 2020. Professora responsável: Kalinka Regina Lucas Jaquie Castelo Branco.

São Carlos - SP
2020

1 Introdução

O conceito de *multithreading* é fundamental para os sistemas operacionais modernos e está intimamente relacionado à multiprogramação. No presente relatório, pretendemos descrever o funcionamento geral por trás do jogo *Profitable Dungeons*, a ser implementado como parte da disciplina *SSC0140 - Sistemas Operacionais* com o intuito de nos possibilitar treinar o uso de *threads* e *semáforos*.

O jogo foi implementado usando-se a linguagem de programação *C++*. Para a implementação da parte gráfica, foi utilizado a ferramenta de interface gráfica Qt.

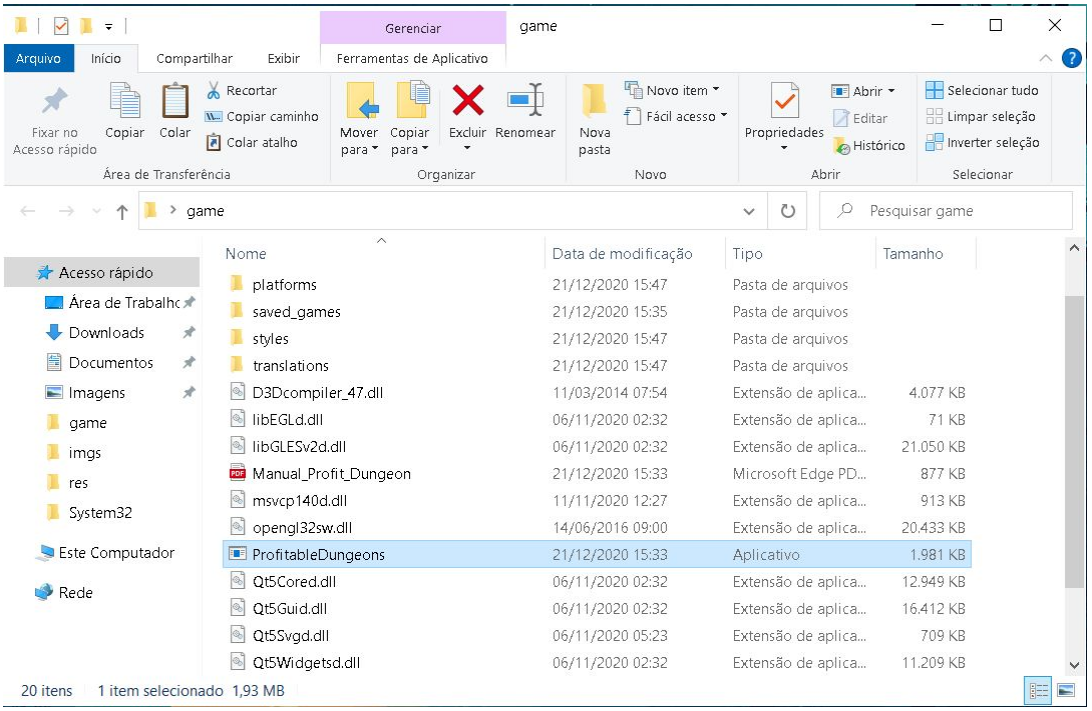
2 Descrição do jogo

O jogo foi desenvolvido baseando-se em um estilo de *incremental game* (influência do *Adventure Capitalist*). O jogador é um aventureiro recém-chegado em uma cidade. Inicialmente pobre, ele poderá investir seu dinheiro comprando negócios, como uma taverna, uma mina, uma loja de roupas, uma loja de alimentos, um castelo e muito mais.
Os negócios do aventureiro podem ser melhorados para que lhe renda mais ouro em menos tempo.

O objetivo do jogo é fazer com que o aventureiro acumule o maior tesouro possível durante a sua vida. Para tanto, o jogador deverá ser capaz de realizar investimentos inteligentes para alcançar a maior quantia de dinheiro possível.
É importante ressaltar que optamos por cortar a parte do jogo focada em RPG e combate em função da falta de tempo para o desenvolvimento.

3 Como instalar

Basta baixar o arquivo game.rar, descompactar e abrir a pasta game, dentro dela terá o arquivo executável *ProfitableDungeons*, como mostra a imagem abaixo, basta abri-lo para jogar o jogo.



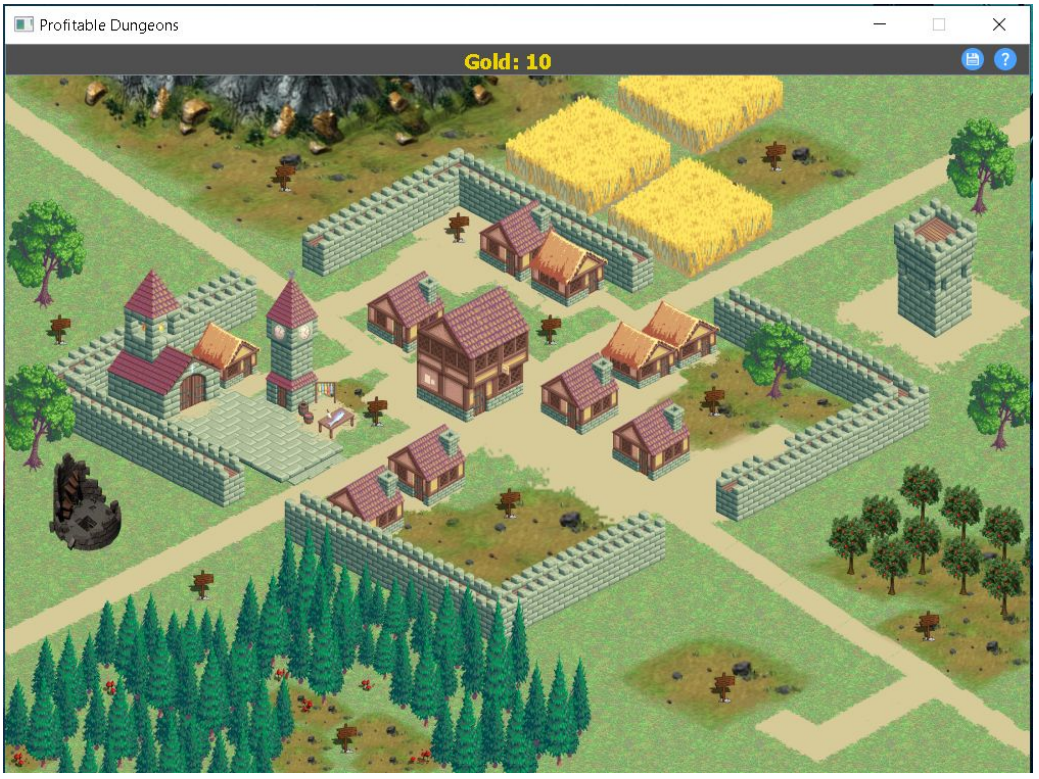
4 Como jogar

Início:

Principais características:

- New Game: será iniciado um novo jogo do começo e o jogador será levado para a tela do jogo.
- Load Game: será pedido que o jogador indique o arquivo do jogo salvo (no formato .pd) e o jogador é levado para a tela do jogo com as informações salvas.
- Quit: Fecha a janela e encerra o jogo.

Tela do jogo:

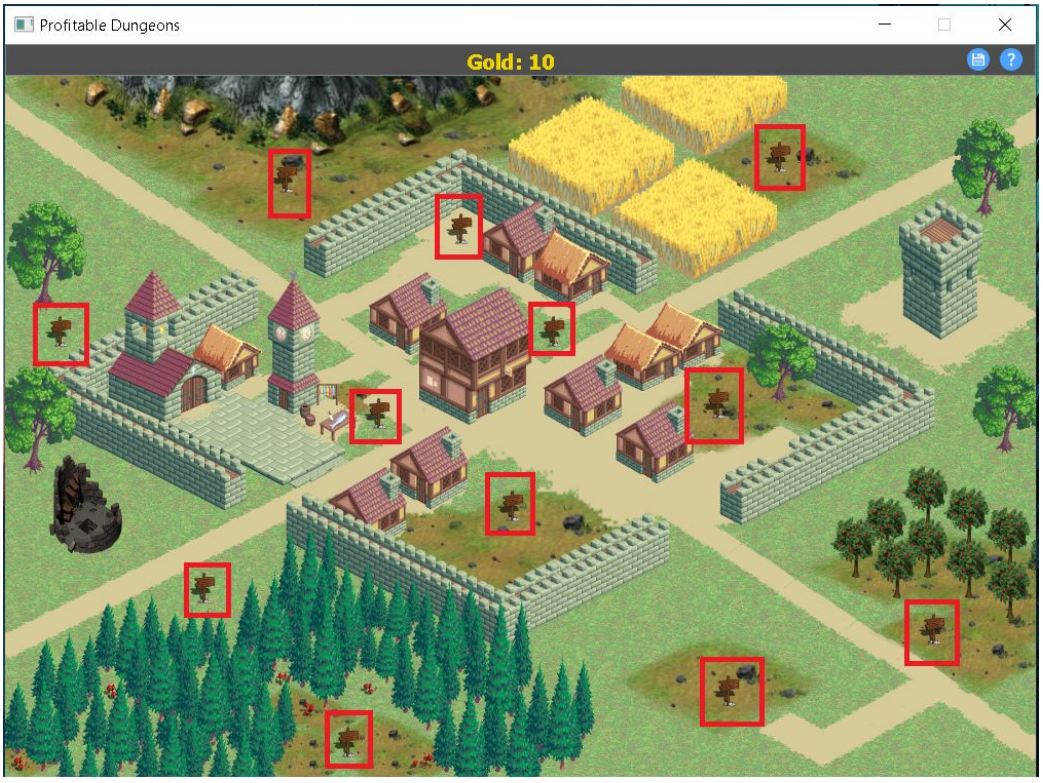


Principais características:

- Gold(centralizado na parte superior da tela): Dinheiro do jogador.
- Placas(espalhadas pela tela): Construções que podem ser comparadas pelo jogador
- Save(símbolo de disquete no canto superior direito): Salva o progresso do jogador
- Interrogação(símbolo de interrogação no canto superior direito): Leva para o pdf do manual do jogo.

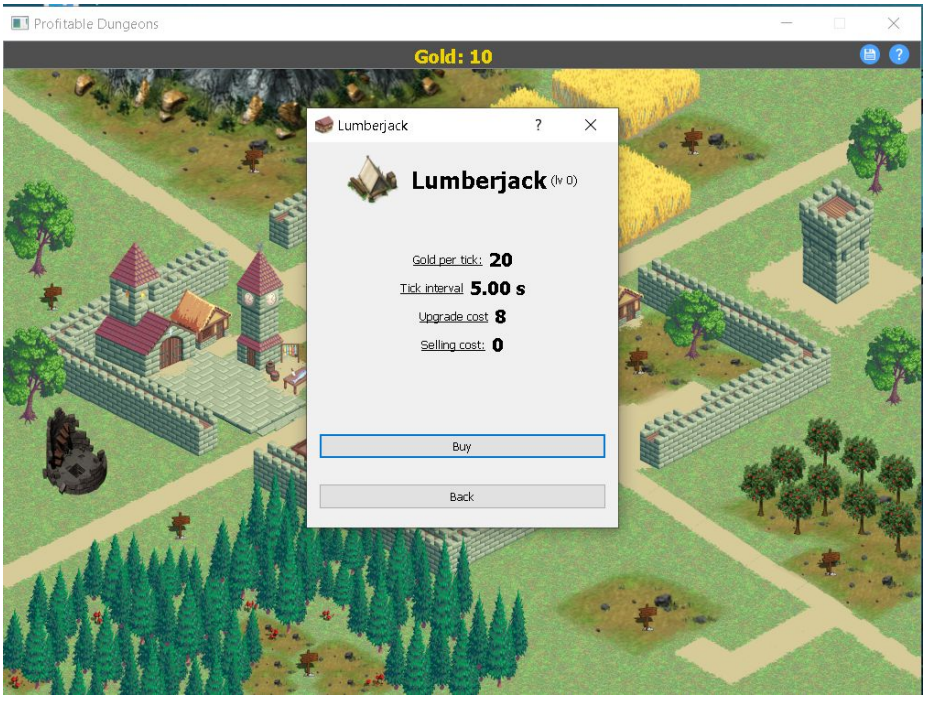
Construções:

Ao iniciar o jogo existem 12 plaquinhas espalhadas pelo mapa, cada uma representa uma construção que pode ser comprada pelo jogador, basta clicar na plaquinha para ver os detalhes de cada construção

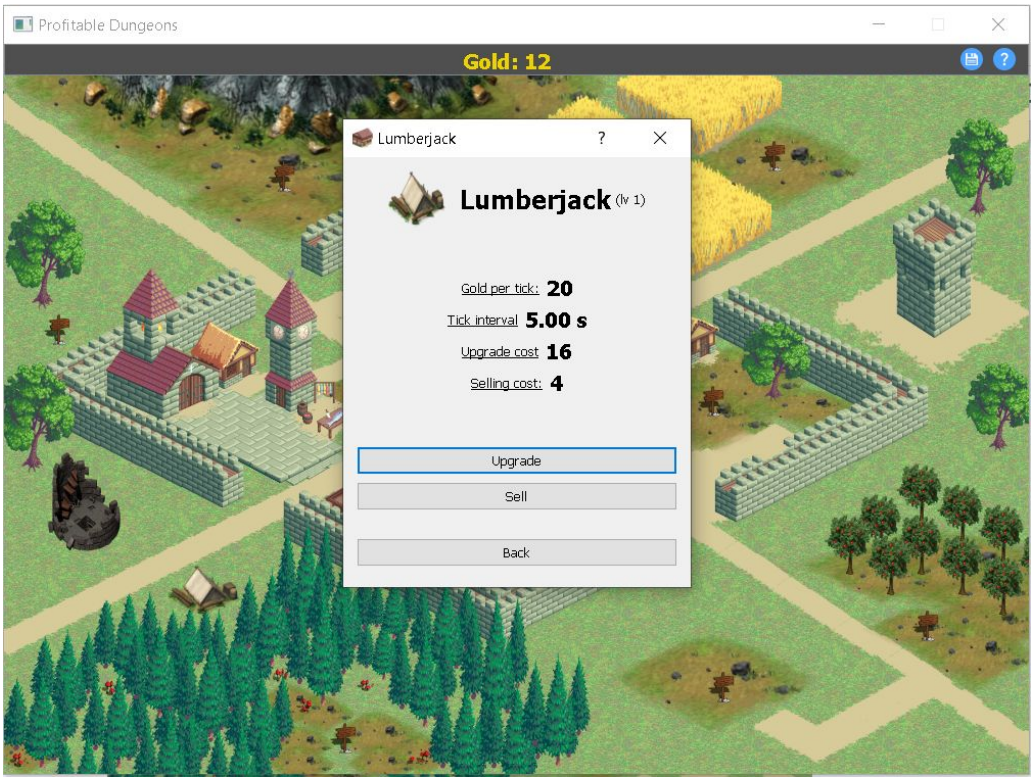


A única construção que começa já construída é a taverna(porque a galera gosta de beber), porém ela ainda está lv 0 e não gera dinheiro para você ainda, é necessário comprá-la como qualquer outra construção. A taverna é o prédio central do mapa.

Tela das construções:
Aparece ao clicar em uma plaquinha ou construção já pronta, se clicar em uma construção ainda não comprada aparecerá essa janela com a opção de compra:



Ao clicar em uma construção já comprada aparecerá essa tela com a opção de upgrade e de venda:

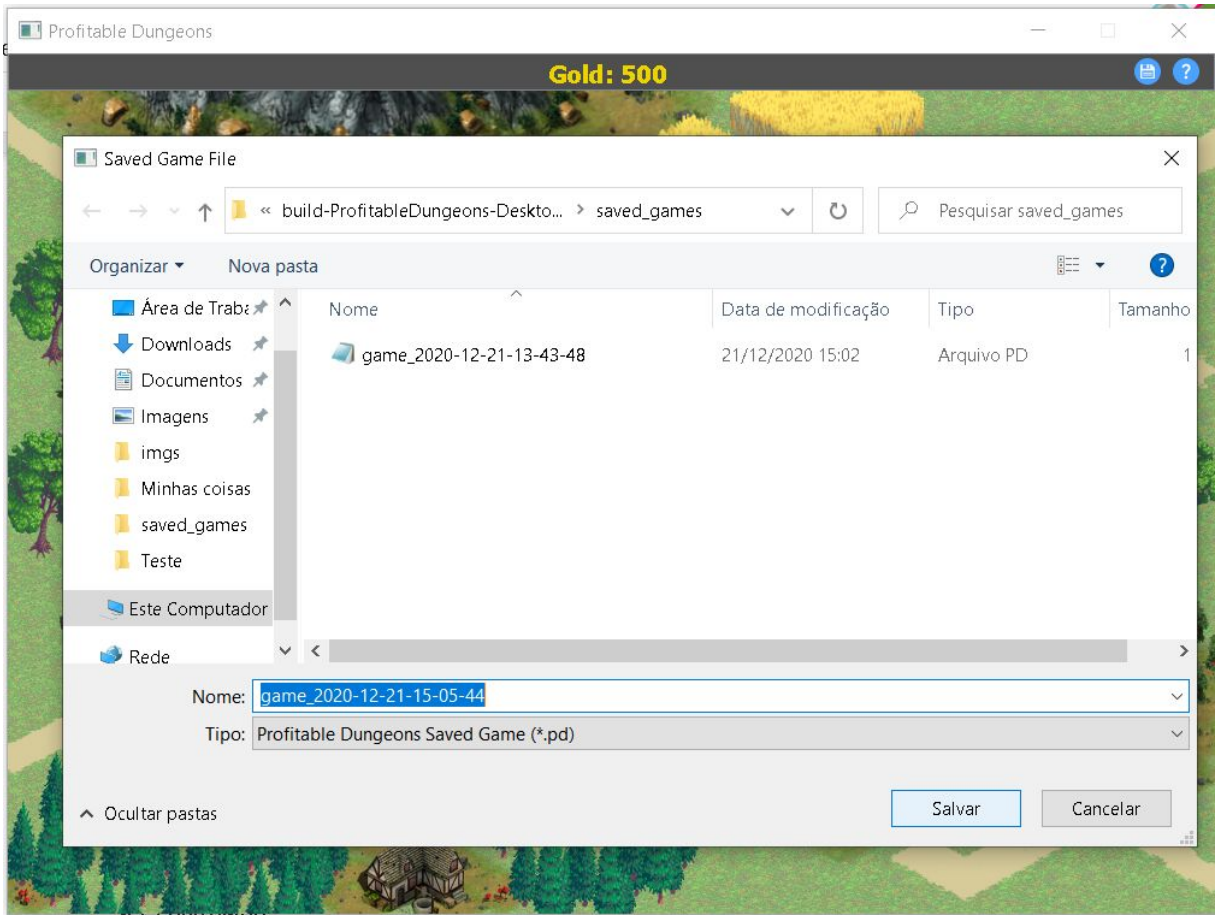


Principais características:

- Nome da construção.
- Level da construção (ao lado do nome).
- Foto da construção.
- Gold per tick: Gold obtido por cada tick.
- Tick interval: Tempo em que ocorre cada tick.
- Upgrade cost: Valor para fazer um upgrade na construção.
- Selling cost: Valor obtido ao vender a construção.
- Buy: Compra a construção
- Upgrade: Compra o upgrade da construção.
- Sell: Vende a construção.
- Back: Fecha a tela da construção e retorna para a tela principal do jogo.

Salvar o jogo:

Ao clicar no botão de save(símbolo de disquete no canto superior direito) é pedido ao jogador que escolha um local para salvar o jogo, basta o jogador escolher onde deseja salvar o save e clicar em salvar, conforme abaixo:



5 Threads

Utilizamos diferentes *threads* para processar cada construção do jogador na cidade. Como cada construção gera uma certa quantidade de dinheiro em um certo intervalo de tempo, elas irão provocar mudanças constantes no dinheiro que o jogador possui em um dado momento. Ao representarmos o trabalho de cada uma das construções do jogador como um *thread* diferente, tornamos a implementação do jogo mais limpa e possibilitamos que a produção de ouro ocorra de maneira mais independente entre as construções.

Com diversos *threads* independentes modificando, constantemente, a quantidade de ouro do jogador (armazenada em uma variável), fez-se necessária a utilização de alguma técnica de sincronização. Caso isso não fosse feito, ocorreriam, frequentemente, condições de corrida entre os *threads*, o que acarretaria comportamento indefinido quando ouro fosse adicionado ou removido do jogador.

6 Semáforos

O problema foi solucionado por meio da utilização de um semáforo. Semáforos são

um tipo abstrato de dados usado para controlar o acesso a recursos compartilhados por múltiplos processos ou *threads*, evitando condições de corrida. Em nosso caso, o recurso compartilhado é a variável que armazena o ouro do jogador. Em nossa implementação, encapsulamos o acesso à essa variável por meio de uma classe chamada *GoldPurse*, ou bolsa de ouro, em português. Essa classe possui, como variável membro, um semáforo, que entra em ação sempre que algum *thread* tenta adicionar ou remover ouro da bolsa. O semáforo é responsável por garantir que apenas uma *thread* atue, por vez, na bolsa de ouro.

Isso se torna ainda mais importante quando consideramos que a modificação no ouro do jogador não se trata apenas de uma alteração no valor armazenado em uma variável, mas também provoca uma nova chamada para redesenhar a porção da interface gráfica que mostra o ouro disponível ao jogador. A implementação da classe Semáforo foi feita por nós mesmos, utilizando apenas ferramentas da biblioteca padrão do C++.

A classe possui uma variável chamada *count*, responsável por indicar quantos recursos há disponível em um dado instante. Em nosso caso, o valor máximo que essa variável assume é 1, visto que o único recurso gerenciado pelo semáforo é a bolsa de ouro. Uma outra variável do tipo *mutex* serve como uma fechadura, permitindo que apenas um *thread* acesse o semáforo por vez. Por fim, uma variável do tipo *condition_variable* é usada para gerenciar o bloqueio e o desbloqueio dos *threads* que acessam o semáforo.

A classe possui ainda dois métodos: *signal* e *wait*. O método *wait* é chamado por um *thread* quando ele deseja acessar o recurso administrado pelo semáforo. Caso o recurso não esteja disponível, o *thread* é bloqueado até que possa acessá-lo. O método *signal*, por sua vez, é chamado por um *thread* que terminou de usar o recurso. Ele irá notificar algum dos *threads* bloqueados que o recurso está disponível.

As chamadas de ambos os métodos são feitas dentro da classe *GoldPurse*, encapsulando as operações de sincronização do semáforo, criando uma abstração a fim de facilitar a adição ou remoção de ouro pelos chamadores.