Novo Tetris UEFS: Uma Nova Era para o Clássico Quebra-Cabeça

João Marcus R. da Silva

[1]Bacharelado em Engenharia de Computação
Universidade Estadual de Feira de Santana (UEFS)
Av. Transnordestina, s/n, Novo Horizonte, Feira de Santana – BA, Brasil – 44036-900

joao.marcus045@gmail.com

Abstract. *This report disagrees about the process and result of a puzzle game in the interpretive language Python, a proposal leveraged by the internship selection for young people, carried out by the publisher and game developer Blizzard Entertainment, the project involves the development of a version of Tetris , generating motivation for students in computational areas. During planning, conditional structures, \textit{loops}, variables and libraries were used, with a main focus on matrices, many of which were integrated into functions. The applied concepts and error tests were detailed in this document, informing their use in software tasks. Finally, the game experience becomes real, where the same objective is to accumulate points.*

Resumo. *Este relatório discorre sobre o processo e o resultado de um jogo de quebra-cabeça na linguagem interpretativa Python, proposta alavancada pela seleção de estágio para jovens, realizado pela editora e desenvolvedora de jogos Blizzard Entertainment, o projeto envolve à elaboração de uma versão do Tetris, gerando uma motivação para estudantes nas áreas computacionais. Durante a codificação foram utilizadas estruturas condicionais, loops, variáveis e bibliotecas, com foco principal em matrizes, muitas das quais foram integradas em funções. Os conceitos aplicados, testes erros foram detalhados neste documento, informando a sua utilização nas tarefa do software. Por fim, a experiência do jogo se torna real, onde o mesmo objetiva o acumulo de pontos.*

1. Introdução

É inegável que com o passar dos anos, a humanidade sempre se propôs ao entretenimento, buscando se conectar com as inovações que surgiam. Através da evolução da computação e seus componentes, tornou-se possível o desenvolvimento dos jogos eletrônicos, além da contribuição significativa para a aquisição e construção de conhecimento. O físico William Higinbotham na década de 50, criou um protótipo de jogo eletrônico chamado "Tennis for Two", utilizando um osciloscópio como uma espécie de monitor, dois jogadores direcionavam uma bola simulando uma partida de tênis. O físico não se preocupou em patenteá-lo, assim não pôde ser considerado o inventor dos jogos eletrônicos [Grupo EPJ 2024].

Não distante desta realidade, alguns anos depois, durante a antiga União Soviética, o cientista russo Alexey Pajitnov, desenvolveu o jogo Tetris, este simulava um quebra-cabeça onde o jogador deveria formar linhas horizontais para adquirir pontos. Resultado disso foi um grande conflito para obter os direitos da sua criação, que teriam sido

roubados, a empresa desenvolvedora de jogos Nitendo reconheceu seu trabalho, implementando a sua criação ao “Game Boy”, se tornando patrimônio cultural, vendendo mais de 35 milhões de unidades e fundando a “Tetris Company”. Assim, o mercado global de jogos ganhou grande importância ao longo dos anos, evoluindo de forma consistente junto com o avanço da computação [O Globo 2024]

Partindo desse pensamento, a desenvolvedora de jogos americana Blizzard Entertainment, abriu uma seleção de estágio para iniciantes, que consiste no desenvolvimento de uma versão do jogo “Tetris”, onde necessitou de algumas regras para uma melhor avaliação, por exemplo deve haver um tabuleiro com 10 colunas e 20 linhas, além de 7 tipos de peças com diferentes formatos:

- I, (linha reta);
- O, (quadrado);
- T, (forma de T);
- S, (Z inclinado para direita);
- Z, (Z inclinado para esquerda);
- J, (L inclinado para esquerda);
- L, (L inclinado para direita).

Além disso, as peças devem surgir e cair aleatoriamente em uma posição no topo do tabuleiro. As formas devem permitir interação com o jogador, possibilitando que sejam movidas e rotacionadas para encaixá-las, formando linhas horizontais sem lacunas. Quando a linha for completada, ela é removida, e as peças acima descem, acumulando pontos, podendo obter o dobro de sua pontuação com 2 ou 4 linhas ao mesmo tempo. Porém, se o jogador permitir que uma peça fixada atinja o topo do tabuleiro, o jogo termina, e a pontuação é exibida. À medida que a pontuação aumenta, a dificuldade do jogo também cresce, o tempo de queda das peças diminui, acelerando o ritmo e tornando o desafio mais intenso. Além dos requisitos, a editora solicitou à adição de uma bomba, que surge como uma peça comum, mas quando encaixada sua explosão eliminaria as peças ao seu redor, sendo uma forma interessante de atrair mais jogadores para o Tetris. Tomando aspectos técnicos como base, o código em linguagem interpretativa Python, deve ser bem estruturado, apresentando estabilidade, resistência e segurança, prevenindo falhas. Nessa perspectiva, fatores como desenvolvimento, metodologia, referencias, estão descritas neste relatório.

Como forma de resolução deste problema apresentado, algumas perguntas necessitam de resposta: 1) Como criar e apresentar o tabuleiro e os formatos; 2) Como gerar as peças e a posição aleatoriamente dentro do tabuleiro; 3) Como movimentar e rotacionar as peças no tabuleiro; 4) Como fazer a verificação de colisão para cada formato; 5) Como simular a gravidade das peças; 6) Como inserir e explodir a bomba no jogo; 7) Como remover linhas e computar a pontuação do jogador; 8) Como determinar o término do jogo e apresentar as pontuações .

- Soluções apresentadas para as questões:

1. É criado uma função principal no código;
2. Nesta função é criado um *loop* principal do programa;
3. Mais funções são criadas para cada execução específica;
4. No *loop*, está alocado a maioria das outras funções;

5. São utilizadas matrizes para a criação do tabuleiro e as peças;
6. No tabuleiro, valores são substituídos pelos formatos do jogo;
7. As funções do código estão conectadas a matriz principal;
8. No *loop*, ocorre a verificação de fim de jogo;
9. Os resultados são apresentados após o usuário perder;
10. O *loop* principal acaba se o jogador não reiniciar a partida;
11. Se reiniciada, o *loop*, a pontuação e o tabuleiro são zerados.

2. Desenvolvimento

Por meio deste tópico, as soluções e a elaboração serão apresentadas, levando como base as questões anteriores levantas durante a introdução.

Como criar e apresentar o tabuleiro e os formatos?Durante a maioria do desenvolvimento do código, como parte principal, se fez necessário o conceito de matriz, a mesma pode ser imaginada como uma ”tabela” feita de listas dentro de outra lista principal. No caso desde programa cada linha da tabela é representada por uma lista separada, e cada valor dentro de uma dessas listas representa um “quadrinho” do tabuleiro, o mesmo possui 20 linhas de 10 colunas [Python Software Foundation 2024]. Não diferente disso, o mesmo conceito foi utilizado para a definição da bomba e as peças do jogo, conhecidas também como “Tetromino”, possuem cor diferentes do tabuleiro.

Para tornar-se possível a visualização do tabuleiro e os Tetrominos, foi atribuído a uma função, a impressão do tabuleiro pré-definido, utilizando laços de repetição, como o *for*, que permitiu percorrer toda a matriz, juntamente com o comando *print* e a formatação “, *end*=’ ’ ”que juntos possibilitaram a impressão dos elementos de forma organizada, assim, deu-se início ao tabuleiro vazio, que posteriormente seria preenchido com os valores presentes em cada peça. Para inserir esses elementos na matriz principal, foi necessária outra função , a mesma utilizou, além de loops*, condições como if , para* identificar os elementos de cada formato de peça substitui-las no devido lugar do tabuleiro, atentando-se para não inserir após o limite definido, finalizando a função após retornar a matriz já modificada e a variável “nova_linha” que serão reutilizadas [Python Software Foundation 2024].

Como gerar as peças e a posição aleatoriamente dentro do tabuleiro? Tendo em vista uma das regras do problema apresentado, onde está especificado que a peça deve ser escolhida e posicionada de forma aleatória, se fez necessário, como solução mais apropriada o uso de uma biblioteca já integrada à linguagem Python chamada de *random, a mesma por já fazer parte da linguagem utilizada, exige somente sua importação no programa.* Essa biblioteca permite gerar números aleatórios com o comando *random.randint()*, usando dois parâmetros: início e limite [Python Software Foundation, 2024]. Com base nisso, foi criada uma função para determinar a posição de inserção da peça no topo do tabuleiro, usando o comando citado e uma variável de limite que define o valor máximo possível para a escolha do surgimento da peça. Já se tratando da escolha aleatória dos “Tetronimo”, foi criada uma função que recebe uma lista com todas as peças do jogo, incluindo a bomba. A função usa o comando *random.choice()* para selecionar aleatoriamente um elemento da lista e retornar um único valor, definindo qual peça aparecerá no início do tabuleiro [Python Software Foundation 2024].

Como movimentar e rotacionar as peças no tabuleiro? Assim como a escolha

da peça, para a rotação da mesma, utilizou-se como resposta mais eficaz o uso da versão 2.1.0 da biblioteca numpy, porém esta exige a instalação no terminal do editor do código-fonte, e sua importação, a instalação foi feita a partir do comando "*pip install numpy*". Ela possui uma estrutura ampla e poderosa contendo conceitos matemáticos e manipulações, permitindo diferentes tipos de interações tornando-a muito utilizada [Numpy Documentation 2024]. Também foi necessário o uso da versão 0.13.5. da biblioteca *Keyboard*, ela permite a automação e identificação de interações com o teclado. Além da importação, exige-se a sua instalação, para isso deve ser escrito no terminal da IDE utilizada "*pip install keyboard*", em seguida, já pode ser usada [Anaconda.org 2024].

Por meio da importação dessas bibliotecas, tornou-se possível a criação de função que entrega uma boa funcionalidade do código e interação do jogador através do teclado. A utilização da função *keyboard.is_pressed("up")* da biblioteca *Keyboard* foi crucial, ela permite a captação de pressionamentos de teclas pré-definidas [Anaconda.org 2024]. A lógica consiste em detectar o pressionamento da tecla *up* (seta para cima), que quando capturada utiliza a função *np.rot90( )* da biblioteca Numpy, que permite a matriz trocar linhas por colunas e, em logo após, inverter a organização de uma das dimensões [Numpy Documentation 2024]. Com parâmetros de matriz, peca, coluna e nova_linha recebidos pela função giro e a junção dessas duas bibliotecas, permitiu girar cada formato, que após passar pela verificação de colisão, substitui a peça antiga por outra com diferente rotação, esta será retornada para ser utilizada posteriormente.

Se tratando da movimentação das peças, também foi importante o uso do *keyboard*, onde a interação com o teclado foi implementada através das funções *keyboard.is_pressed('left')* e *keyboard.is_pressed('right')*. A lógica por trás desse código consiste em monitorar as teclas pressionadas para modificar a posição de uma coluna. Se permitido pela verificação de colisão, ao pressionar a tecla esquerda, o valor da coluna é subtraído em 1, enquanto a tecla direita resulta em uma adição de 1 ao valor da coluna. Após cada interação, o código retorna o valor atualizado da coluna, para ser reutilizada. Além disso, na função gravidade, contém a verificação de pressionamento de tecla para baixo, utilizando o *keyboard.is_pressed("down")* que diminue o tempo do *delay*, acelerando o processo da função, permitindo ao jogador a escolha de fazer a peça cair mais rapidamente.

Como fazer a verificação de colisão para cada formato? Afim de melhorar a funcionalidade e evitar falhas que comprometeriam toda jogabilidade do Tetris, surgiu a demanda de verificações, para agrupar a maioria delas foi criado uma função chamada Verificar, esta recebe valores como matriz, peca, coluna, nova_linha, esses dados estão presentes na maioria do programa, podendo resultar em substituições indesejadas, tornando mais do que necessário a utilização da função que verifica colisões, ela contém dois *loop* for, onde existem condições indentadas que monitoram situações, como a parada da peça ao chegar no fim da matriz, se algum tipo de colisão entre peças ou com o próprio tabuleiro foi detectada, que para a simulação de gravidade isso se torna um ponto crucial. Se as condições forem favoráveis retorna-se um valor *booleano True*, tornando possível a interação das peças e tabuleiro presentes no jogo, se não for favorável retorna um valor *booleano False*, interrompendo a próxima ação da peça. É importante falar que a função de verificação está presente em boa parte das outras funções no código, como na movimentação, rotação e gravidade, reduzindo a taxa de falhas durante a execução do

```
#Função que verifica a maioria das interações entre as peças e o tabuleiro
def Verificar(matriz,peca,coluna, nova_linha):

    for l in range(len(peca)):
        descer=True
        for c in range(len(peca[l])):
            if peca[l][c]!=" ":
                #Verifica se é a última linha do tabuleiro
                if nova_linha+l+1>=len(matriz):
                    return False

                #Verifica se é a última linha da peca
                if l+1<len(peca):

                    #Verifica se abaixo da bloco preenchido tem outro bloco da própria peça
                    if peca[l+1][c]!="":
                        #Se não houver bloco, a variável descer armazena o valor e aguarda a próxima verificação
                        descer=True
                    #Se houver, descer armazena False, e aguarda a próxima verificação
                    else:
                        descer=False
                # Verifica se o bloco abaixo do observado abaixo não está preenchido
                if matriz[nova_linha + l + 1][coluna + c] !=" " and descer:
                    return False

    return True
```

Figure 1. Função de verificações.

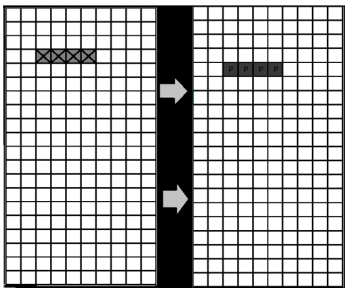

Figure 2. Ilustração da gravidade.

código.

Como simular a gravidade das peças?Para que o programa desenvolvido seguisse o caminho de ser uma versão do jogo Tetris não poderia ser negligenciado a visualização das peças descendo no tabuleiro. Partindo disso, surgiu a necessidade de criar uma função que imitasse a gravidade, tornando importante o uso de duas bibliotecas, que juntas teriam um papel importante na ilusão criada durante a execução do jogo. Para apagar o terminal foi utilizado a função *os.system("cls")* da biblioteca os, esta permite interagir com o sistema operacional, além de executar comandos diretamente no terminal [Python Software Foundation 2024]. Outra função utilizada por todo o código foi *time.sleep( )* da biblioteca time, esta fornece diferentes funções relacionadas ao tempo, a função que foi usada permite criar *delays* e temporizadores durante a execução do código. [Python Software Foundation 2024]. Ambas as bibliotecas já pertencem ao Python exigindo somente a sua importação no programa, porém se tratando da biblioteca os, é importante ressaltar que o comando *os.system("cls")* só funciona para o sistema operacional Windows, para o SO Linux é necessário a modificação do "*cls*" para "*clear*", assim o código retoma sua ação normalmente.

Embora essas funções sejam essenciais para a gravidade, ainda é necessário criar outra tarefa automatizada. Será preciso substituir os valores da peça em sua posição antiga para evitar rastros indesejados no tabuleiro. A função Apagar recebe os parâmetros matriz, peca, coluna e nova_linha, e, por meio de um módulo de verificação, localiza a posição onde o formato está e substitui cada parte dele pelo fundo do tabuleiro.

Enfim, as tarefas automatizadas já definidas fora da função gravidade, ocorre a junção delas para a execução e simulação desejada. Nela a variável nova_linha será sempre zerada no início, além disso, a gravidade possui um loop while responsável por manter

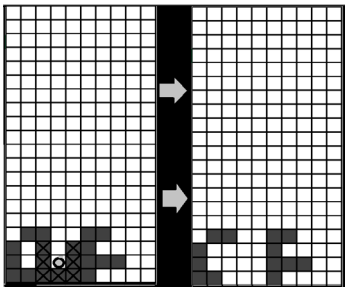

Figure 3. Ilustração da explosão da bomba

a execução das outras tarefas indentadas abaixo, enquanto a verificação presente no laço de repetição retornar o valor True, as tarefas como *Apagar, girar, movimentar*, *ColocarPeca*, *imprimirMatriz, os.system* são recrutadas consecutivamente. Primeiro o laço verifica, apaga a peça na posição atual, em seguida, computa as possíveis modificações geradas pelas funções de movimentação e giro, em uma linha abaixo substitui os valores do tabuleiro pelos valores da peça, imprime o tabuleiro atualizado, gera uma pausa no código e apaga o terminal.

Ademais, afim de gerar uma melhor experiência ao jogador, foi criada uma função para aumentar a dificuldade, que recebe a quantidade de pontos obtidos e ajusta o tempo de execução da gravidade. A cada meta de pontuação alcançada, o delay diminui, tornando o jogo mais desafiador e interativo. Embora a função esteja definida externamente, ela é chamada dentro da função de gravidade para que o valor do tempo seja melhor computado. A função de explosão da bomba também está presente. Ela é chamada no final da função de gravidade, fora da indentação do laço while, pois será acionada apenas quando a verificação do laço retornar o valor False, indicando uma colisão. Isso permite que a bomba execute sua tarefa pré-estabelecida. Além disso, a interação do pressionamento da tecla para baixo também faz parte da função de gravidade, permitindo que o jogador mova as peças mais rapidamente.

Como inserir e explodir a bomba no jogo? Para tornar a ideia da bomba algo concreto, foi necessário cria-la como uma peça qualquer, levando em consideração a sua possível interação com o jogador, geração aleatória e sua colisão, porém ao colidir deve apagar as peças ao seu redor em um formato 3x3.

A forma com que a bomba elimina as peças quando acionada não difere muito da função responsável por apagar as peças, a explosão utiliza um modulo de verificação para encontrar a sua posição substituindo o seu valor e as coordenadas de linha e coluna ao seu redor por cores presentes no fundo do tabuleiro. Esta mesma função possui verificações específicas, evitando a destruição do limite de colunas do tabuleiro, se acontecer da bomba colidir com uma peça no início do tabuleiro, a explosão não ultrapassará o limite de linhas da matriz, se não houvesse essa verificação, isso não seria possível fazendo com que a explosão substituísse valores do lado oposto da matriz (no final do tabuleiro). É possível aprimorar este protótipo, pois em alguns casos a verificação falha substituindo partes das peças pela bomba durante sua movimentação, este problema pode ser resolvido por meio de uma melhor verificação, uma que atenda a demanda da bomba.

Como remover linhas e computar a pontuação do jogador? Sabe-se que a

pontuação do jogo é um fator importante, no caso do Tetris, a cada linha horizontal completada obtém-se pontos, se forem 2 ou 4 linhas simultaneamente esses valores são dobrados. Para a remoção de linha, foi utilizado um módulo de verificação que percorre toda a matriz, que ao identificar um fragmento de peça, adiciona o valor 1 em um contador, se este chegar a 10, a linha verificada é removida, e logo após insere-se outra vazia no início do tabuleiro, contador *remove_L* armazena a quantidade de vezes que isso acontece. Com a pontuação sendo pré-estabelecida a 100 pontos vezes a quantidade de linhas, se o contador *remove_L* ser maior que, recebe o dobro de pontuação.

Como determinar o término do jogo e apresentar as pontuações? Assim como o "Tetris" original o fim de jogo é definido quando uma nova peça não pode ser inserida no tabuleiro, o programa feito na linguagem Python atende esse requisito, Através da função principal presente no código, a mesma que agrupa a maioria das outras funções do software, possui um *loop* for que verifica se a alguma peça roxa já fixada na matriz, se isso acontecer o *loop* principal é interrompido, e logo após chama-se a função *FimDeJogo*, esta imprime a pontuação final do jogador e o dá a oportunidade de jogar novamente ou sair, se o jogador pressionar a tecla "P" a matriz e a pontuação são reiniciadas, se ele pressionar "S" ou esperar cronômetro acabar o jogo é finalizado.

3. Manual de uso Testes

3.1. Manual de uso

Para rodar o programa, você precisará ter um interpretador Python instalado em sua máquina. Siga estas etapas para executar o jogo:

1. Inicie o interpretador Python e acesse o arquivo..
2. Execute o código-fonte por meio do interpretador.

No início do jogo, há uma interface simples onde o jogador deve inserir seu nome. Em seguida, o jogo apresenta um tutorial que explica as teclas, o objetivo e outras informações. Após essa introdução, o jogo começa. Ao final de cada partida, o sistema pergunta se o jogador deseja jogar novamente. Se a resposta for afirmativa, o jogo reinicia. Caso o jogador opte por não continuar ou não escolha uma das opções disponíveis, o jogo será encerrado.

3.2. Testes

Levando em consideração o nome "Exemplo", para testes.

Apresentação da primeira interface do programa, onde o usuário irá informar o seu nome, dando início ao jogo. Representada na imagem 4.

Nota-se na imagem 5 que por meio da interface apresentada, o usuário poderá se informar sobre o objetivo e funções do jogo.

Considerando o fim de partida causado pelo amontoado de peças que tocam o topo, a imagem demonstra oque acontece a seguir. Observa-se na imagem 6 que o usuário possui duas opções, essas determinam se o jogador irá reiniciar a partida ou sair do jogo.

Figure 4. Interface inicial.

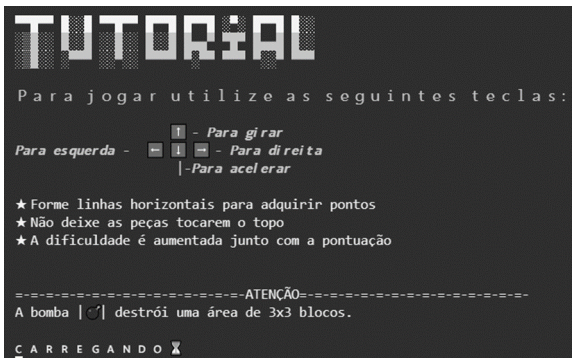

Figure 5. Interface do tutorial.

Figure 6. Interface do fim de jogo.

3.3. Erros

Em relação aos erros encontrados no código, o problema mais comum ocorre durante a inserção da bomba. A verificação falha e acaba substituindo partes das peças pela bomba enquanto ela se move. Esse problema pode ser resolvido por meio de uma verificação mais robusta, que leve em conta as características específicas da bomba. Ademais, em algumas situações, uma peça pode ”engolir” partes de outra ao girar no topo do tabuleiro e cair sobre elas. Além disso, não foi possível fazer a peça I girar no eixo da mesma, ela gira a partir do primeiro quadrado.

É importante esclarecer que durante os testes, não ocorreu erros que poderiam interromper a execução do código, apenas algumas falhas de verificações que atrapalham a experiência do jogador.

4. Conclusão

Este relatório exibiu uma resolução computacional em Python, uma linguagem interpretativa, para uma versão do jogo Tetris. A abordagem desenvolvida resolve o problema proposto utilizando métodos e estruturas de dados, como listas, bibliotecas, matrizes, e *loops*, para armazenar e verificar interações do jogador. O projeto inclui funções que dividem as tarefas em blocos, facilitando a compreensão do código e tornando a codificação mais organizada e modular. Além disso, a solução aprimora a dinâmica do jogo antigo, tornando-o mais interativo e atraente por meio de novas jogadas ,funcionalidades e menus. No entanto, o sistema apresenta erros que podem surgir verificações incorretas, estas atrapalham parte da jogabilidade. Possíveis modificações no futuro poderão melhorar o tratamento desse tipo de falha, implementando um sistema de verificação que limite interações indesejadas. Isso resultará em um código com menos falhas de execução, proporcionando uma experiência ao jogador.

5. Referências

NUMPY. Como instalar o NumPy. Disponível em: https://numpy.org/pt/install/. Acesso em: 28 out. 2024.

NUMPY. Numpy Documentação. Disponível em: https://numpy.org/doc/stable/. Acesso em: 28 out. 2024.

ANACONDA. Biblioteca Keyboard. Disponível em: https://anaconda.org/conda-forge/keyboard. Acesso em: 28 out. 2024.

GITHUB. Documentação Keyboard. Disponível em: https://github.com/boppreh/keyboardapi. Acesso em: 28 out. 2024.

GRUPO EPJ. Videogame: Uma jornada épica pelo universo dos jogos eletrônicos. Disponível em: https://epraja.com.br/videogame-uma-jornada-epica-pelo-universo-dos-jogos-eletronicos/. Acesso em: 28 out. 2024.

CAUSIN, Juliana. Tetris: Origem do jogo russo ’invencível’ de 1984 que foi dominado por um adolescente nos EUA. Disponível em: https://oglobo.globo.com/economia/tecnologia/noticia/2024/01/05/tetris-conheca-a-origem-do-jogo-russo-invencivel-de-1984-que-foi-dominado-por-um-adolescente-nos-eua.ghtml. Acesso em: 28 out. 2024.

PYTHON SOFTWARE FOUNDATION. Python documentation. Disponível em: https://docs.python.org/3.12/. Acesso em: 28 out. 2024.