Candidato: Felipe Zarattini Miranda  
Data: 06/08/2023  
  
Assets utilizados ao longo da implementação do teste:

Sirenix Odin Inspector

Doozy UI Manager

Probuilder

Dreamteck Splines

TextMesh Pro

Assets visuais grátis da Unity Asset Store: Low Poly Fruits e DragonKnife

O projeto foi desenvolvido ao longo de 3 dias com duração de 3 a 6 horas em cada dia.

**1º dia:**

Após jogar o jogo original e fazer uma análise do que seria necessário para desenvolvê-lo, decidi fazer uma experimentação com Spline do pacote da Dreamteck, para poder implementar mais rapidamente a movimentação da faca e ter mais controle sobre ela. Implementei um pequeno protótipo para fazer o teste, mas não gostei do game feel, pois a movimentação não ficou natural. Talvez com utilização de curvas de aceleração o resultado tivesse ficado melhor.  
 Decidi mudar a implementação e utilizar a própria física do Unity através de Rigidbodies e colisores. Com a física implementada, comecei a pensar em qual seria a melhor forma de dividir os meshes dos objetos cortáveis ao meio.   
 Conheço um método para cortar um mesh utilizando um plano, o que permitiria cortar o mesh em qualquer direção e sentido, ou cortar objetos que ficassem se mexendo na tela (caso os objetos ficassem rodando em cima das plataformas). Como não conseguiria implementar este algoritmo sem referências ou códigos prontos no tempo proposto, decidi seguir pelo caminho mais simples e dividi os objetos utilizando o Probuilder de forma que as partes esquerda e direita já fossem pré-separadas como filhos do objeto principal.

  
 Criei SriptableObjects para guardar dados relevantes tanto da faca quanto dos objetos cortáveis (CutterData e CuttableData).  
  
**2º dia:**  
 Implementei os tratamentos das colisões dos objetos do jogo. Ao invés de realizar todos os tratamentos nas partes da faca (lâmina e punho), decidi que cada objeto ficaria responsável pelas suas próprias ações ao detectar uma colisão. Desta forma:  
   
**Lâmina:** Ao detectar uma colisão decide se a faca ficará presa ou não.  
**Punho:** Ao detectar uma colisão decidirá se irá quicar ou não.

**Objetos cortáveis:** Ao detectar uma colisão decide se será cortado ou não.  
**Área de fim de jogo:** Envia um evento de fim de jogo (derrota) ao detectar uma colisão com o jogador.  
**Área de pontuação:** Pontua e envia um evento de fim de jogo (vitória) ao detectar colisão com o jogador.  
 Para facilitar as detecções nas colisões um ScriptableObject chamado TagData contendo as tags dos objetos do jogo está referenciada em scripts com tratamento de colisão.  
 A maior dificuldade do desenvolvimento foi lidar com bugs relacionados a física do próprio Unity quanto a essas colisões. O mais estranho de todos foi um em que a OnTriggerEnter estava sendo chamada duas vezes (uma para o colisor do objeto pai e uma para o colisor do objeto filho). Este problema estava fazendo com que a pontuação do jogador fosse dobrada.  
 Este é um problema clássico facilmente resolvido utilizando booleanas de controle para garantir que a colisão seja processada apenas uma vez. Acredito que por algum problema interno com o cache do Unity esta solução não estava funcionando (verifiquei com break points que a booleana nunca era alterada). O problema foi resolvido após fechar e abrir o Unity.

**3º dia:** Terminei de construir as duas fases do jogo, e terminei de implementar o fluxo através dos levels que havia começado no dia anterior. Implementei também a UI mostrando a pontuação do jogador e as telas de vitória e derrota. O jogo segue por duas fases. Caso o jogador morra em uma delas pode refazê-la. Ao chegar ao final da última fase, ele pode recomeçar o jogo inteiro.