

**JACK DANCE’S**

Game Design Document

Versão: 1.0

**Autores:**

F018989 – Guillermo de Camargo A. Benitez

N439DJ9 – Lucas Reghini Magalhães

São Paulo,

junho de 2019

Índice

[**1.** **História** 3](#_Toc9017731)

[**2.** **Gameplay** 4](#_Toc9017732)

[**3.** **Personagens** 5](#_Toc9017733)

[**4.** **Inimigos** 6](#_Toc9017734)

[**5.** **Controles** 7](#_Toc9017735)

[**6.** **Câmera** 8](#_Toc9017736)

[**7.** **Cutscenes** 9](#_Toc9017737)

[**8.** **Melhorias Futuras** 10](#_Toc9017738)

[**9.** **Desenvolvimento** 11](#_Toc9017739)

[**I.** **Telas do Jogo (aba Design)** 11](#_Toc9017740)

[**II.** **Telas do Jogo (aba Bloco)** 17](#_Toc9017741)

[**10.** **Publicação** 29](#_Toc9017742)

[**I.** **Detalhes do APP** 29](#_Toc9017743)

[**II.** **Recursos Gráficos** 30](#_Toc9017744)

[**III.** **Classificação de Conteúdo** 32](#_Toc9017745)

[**Requisitos de Entrega** 34](#_Toc9017746)

1. **História**

Em uma cidade grande havia um homem que adorava frequentar baladas e era um grande fã de whisky's bourbon. Ele era tímido, porem ao beber do seu elixir ele encarnava poderes cênicos, assim com grande destreza ele executava passos nunca visto, Jack dançava!

Porem devido a seus excessos, seus passos de dança eram prejudicados, fazendo-o assim perder o controle e entrando em desmaio, assim encerrando seus poderes.

Jack Dance's é um jogo de memória, oferecendo para o jogador a possibilidade de testar sua memória de uma maneira divertida com passos engraçados do Jack alcoolizado na balada.

Junto com Jack o jogador deve acertar todas as cores em ordem, para manter o Jack dançando.

1. **Gameplay**

Em Jack Dance's a mecânica consiste em um árcade com o objetivo de acertar as cores que apareceram para o jogador (no globo de espelhos e chão de led) em ordem, sem errar a cor. Para escolher a cor basta clicar nos círculos coloridos que correspondem à cor da ordem.

O grande desafio do jogador é sua própria memória, como Jack Dance's é um jogo de memória, o objetivo do jogador é acertar a ordem das cores, é a maneira para evoluir no jogo e melhorando a memória do próprio jogador, como se o usuário evoluísse, adaptando sua mente para alcançar uma melhor pontuação.

O jogador avança naturalmente enquanto joga, conforme vai acertando as cores as sequencias vão ficando maiores, assim aumentado o desafio e exigindo mais da memória do jogador.

A gameplay se relaciona com a história de Jack, pois ele adora dançar após ingerir seu elixir.

A recompensa vem através de ranking, pois quanto mais o jogador acerta a sequência de cores maior será sua pontuação, assim ele pode salvar a sua pontuação com seu nome no ranking. O objetivo é se superar ou superar alguém que fez uma pontuação superior.

O jogador não recebe uma vitória literal, pois o jogo não tem fim, por ser em estilo arcade o jogo prossegue até o jogador errar a sequência da cor.

Para perder em Jack Dance's basta errar uma cor da sequência.

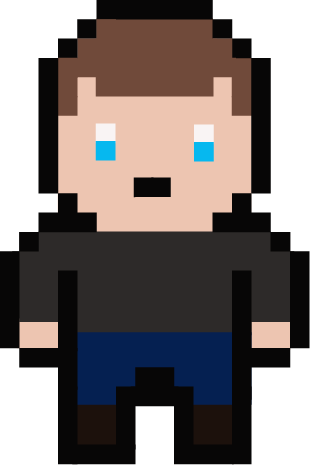
1. **Personagens**

Jack é um homem de 25 anos, olhos azuis, cabelo castanho claro que é apaixonado por whisky's, principalmente os Bourbons (que contem ao menos 51% de milho em sua composição). Ele é conhecido em todo os lugares que passa, pois, sua dança fica na cabeça de todos que já o assistiram.

Jack sempre foi uma pessoa tímida, porem ao consumir muito de seu whisky favoritoperde toda a sua timidez e começa a dançar.

Jack é tímido, porem somente quando está sóbrio.

O grande poder de Jack é poder dançar de maneira impressionante e inesperada somente ao consumir seu elixir “secreto”.

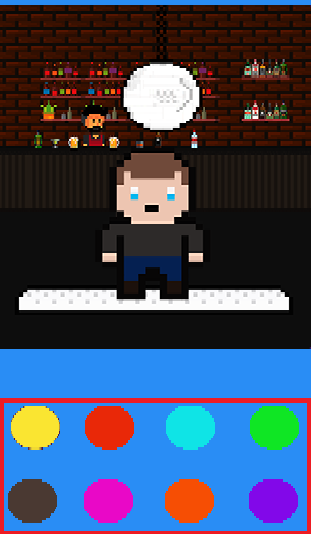
****

**Jack pode dançar**

1. **Inimigos**

Em Jack dance's não existe inimigos, somente a memória do jogador que pode falhar.

1. **Controles**



Podemos observar na imagem a cima, o sistema de controle do jogo de dá pelos botões coloridos dentro do quadrado vermelho da imagem. Cada um representa uma das cores que podem aparecer conforme Jack dança, como o objetivo é que o jogador repita esta sequencia de cores os botões foram dispostos dessa forma.

1. **Câmera**



O jogo tem uma câmera orientada para o 2D de frente.

Arte no estilo Pixel Art, utilizando mudança de imagens para criar efeitos de movimento.

1. **Cutscenes**

O jogo não contem Cutscenes.

1. **Melhorias Futuras**

Implementar uma versão PvP, onde no mesmo celular seja possível enfrentar seus amigos como oponentes, tanto online quanto local revezando o celular, assim um joga de cada vez, repetindo a sequencia anterior e adicionando no final uma nova cor.

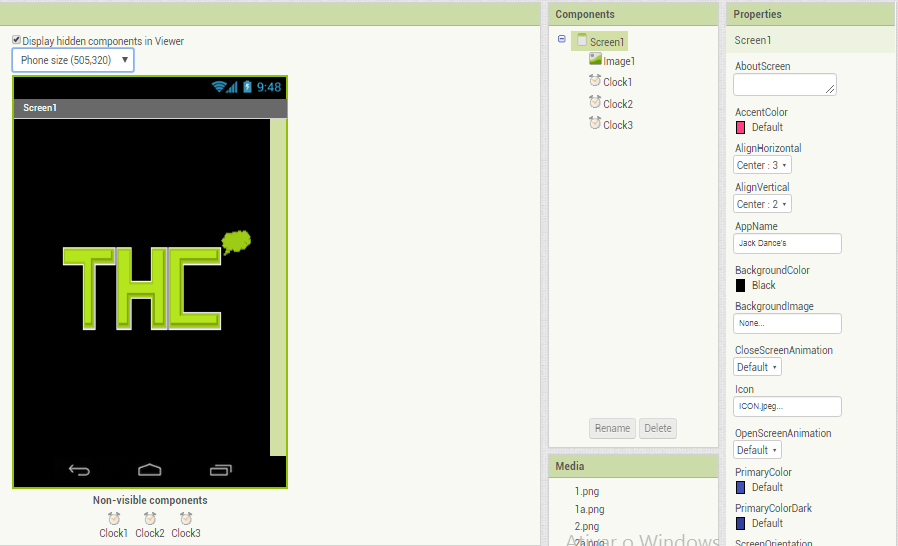
Melhorar a lógica de programação, como está é uma primeira versão, ainda é possível melhorar partes do código, eliminando trechos, criando rotinas para serem executadas ao invés de processos repetidos.

Adicionar mais sprites para Jack ter mais danças.

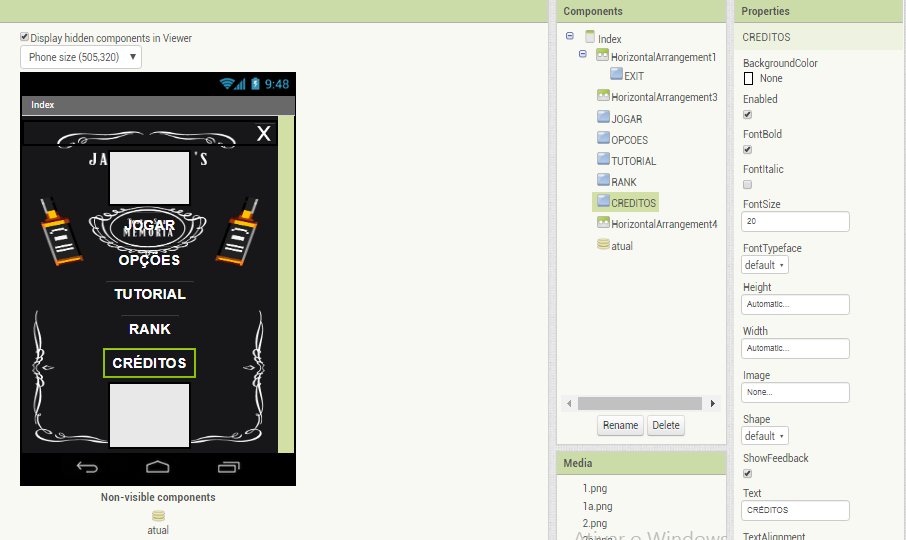
Adicionar um Ranking online mundial.

Colocar uma trilha sonora, como o jogo se passa em um ambiente com animado e com dança, uma trilha sonora seria extremamente interessante.

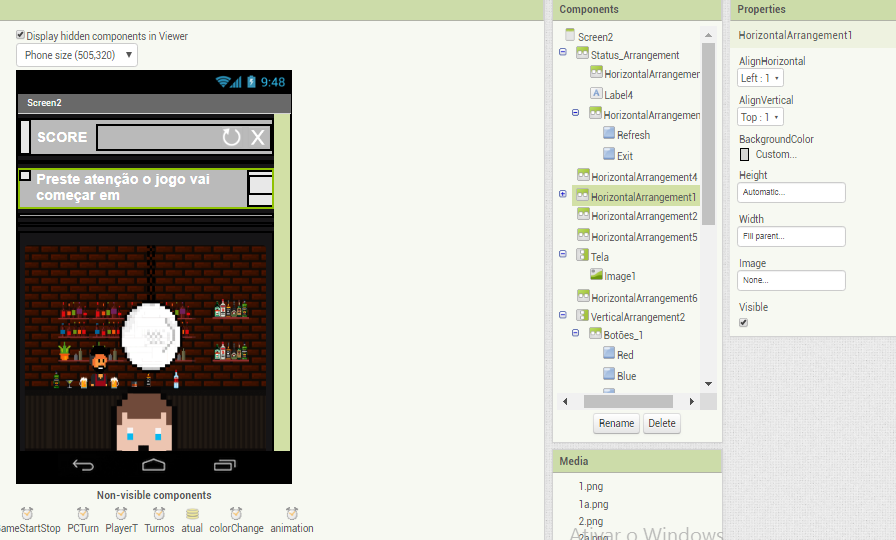
1. **Desenvolvimento**
   1. **Telas do Jogo (aba Design)**



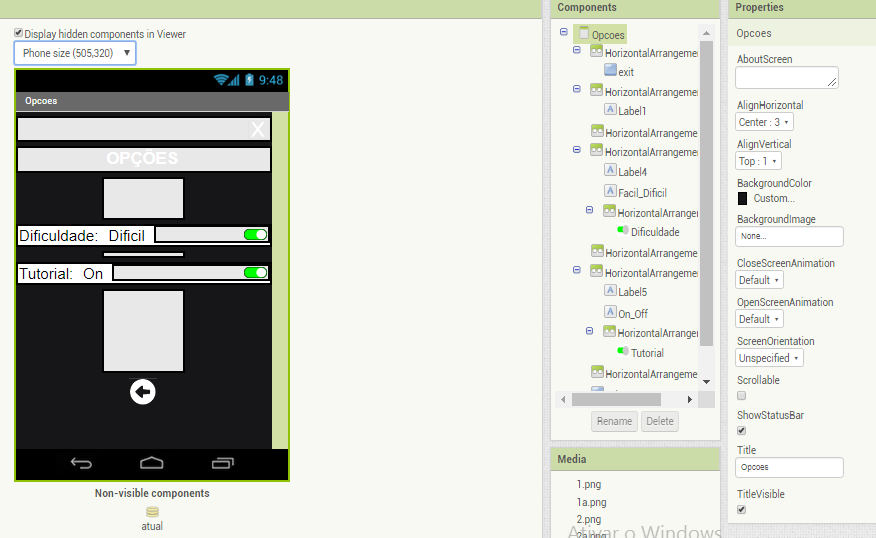
Screen1 – É a *Splashscreen*, nela temos os logos da companhia, Unicluster e Unip. Composta por 1 Imagem e 3 Relógios.



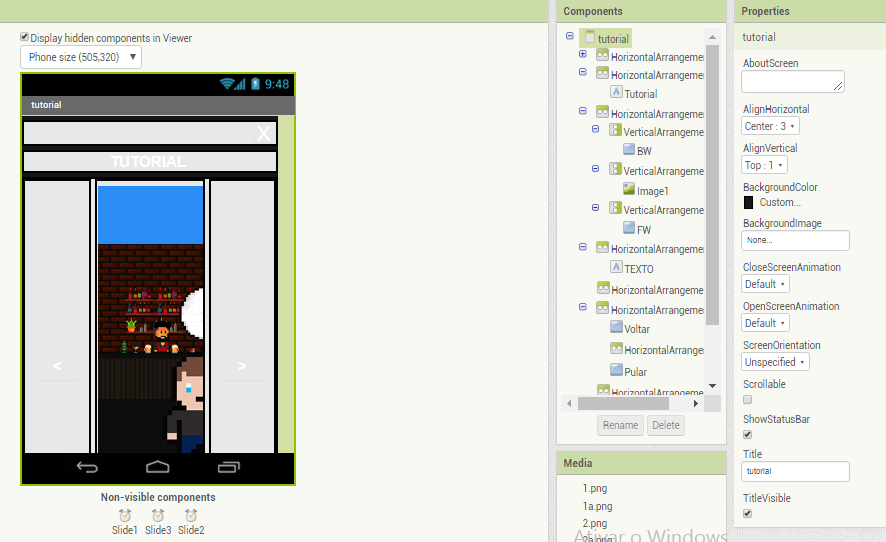
Index – É a tela principal do jogo, da acesso as outras páginas pelo menu de Botões. É composta por um Botão “EXIT’’ que finaliza o aplicativo. Também temos outros 5 botões que formam o menu principal dando acesso a outras telas referentes aos próprios nomes dos Botões. Por ultimo temos um TinyDB nomeado atual.



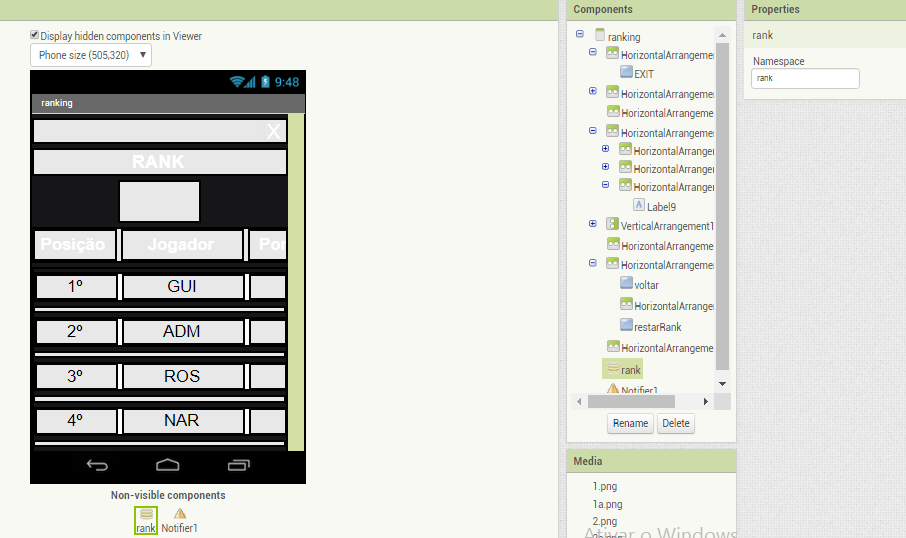
Screen2 – Está tela é onde o jogo em si é executado. Na parte superior temos alguns *Labels* E Botões com imagens, está área é responsável por passar informações ao jogador e oferecer comandos ‘’importantes’’ como fechar o aplicativo ou reiniciar o jogo. Em seguida temos uma Imagem, ela será trocada de acordo com o jogo para dar efeito de dança ao personagem e mudar as cores. Na parte inferior temos os dois arranjos, cada um com 4 botões utilizados para interagir com o jogo, eles não têm nada escrito, são visualmente compostos apenas pelas imagens. Por último temos um *TinyDB* utilizado para salvar o score e verificar a dificuldade do jogo, e diversos Relógios utilizados para desenvolver o jogo.



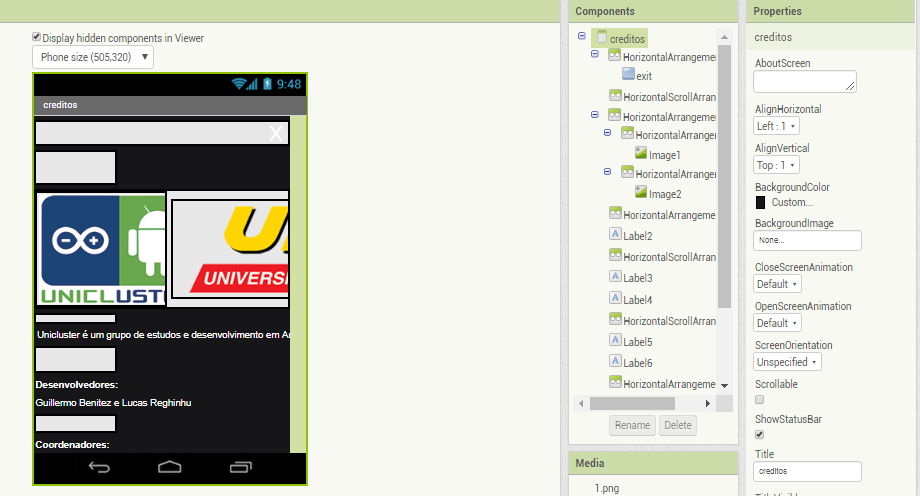
Opções – Esta tela apresenta algumas opções do jogo. Como todas tem um botão “EXIT”, um *Label* com o título da tela, e depois dois arranjos iguais com um dois *Labels*, um com a o nome da opção e outro com seu estado, seguidos por um *Switch* utilizado para controlar a opção. No final da tela temos um botão para voltar a tela “Index”, e um *TinyDB* utilizado para salvar as opções.



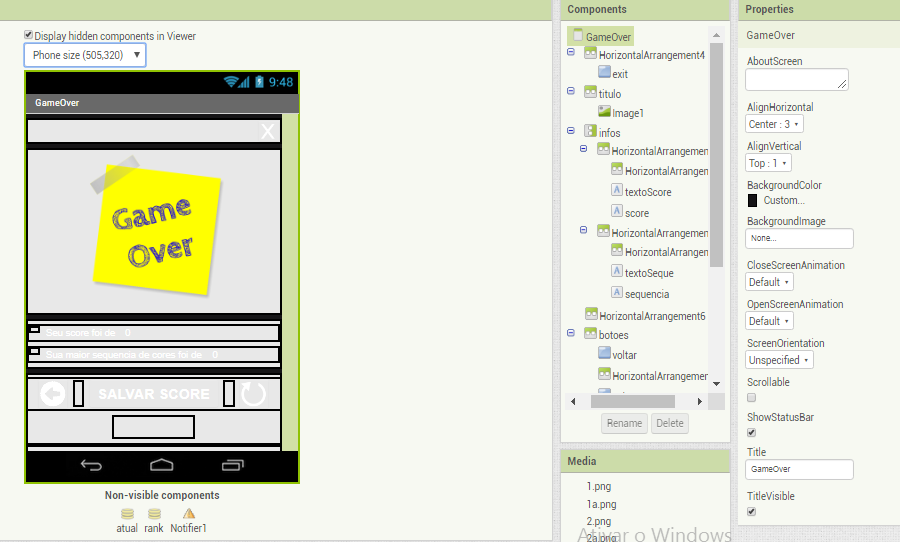
Tutorial – Como o próprio nome diz, está tela serve para explicar como funciona o jogo. Ela tem o cabeçalho padrão (“EXIT” e *Label* Título) seguido por 3 arranjos, os das laterais contem apenas um botão cada, responsável por passar os “slides”, enquanto o central tem um imagem que pode ser trocada dentro do próprio “slide” para dar efeito de animação, quanto quando trocar de “slide” em si, para ajudar a passar a próxima mensagem. Abaixo encontramos um *Label* onde será apresentado a informação escrita e no fim da tela dois botões, um para voltar ao “Index” e outro para pular direto para a tela “Screen2” e jogar o jogo. Por último temos 3 relógios utilizados para as animações dos “slides” pertinentes.



Rank – Está tela apresenta o *rank* com os 5 melhores recordes de pontuação. Temos o cabeçalho padrão seguido por arranjos que compõem o *rank* em si. O primeiro é composto por 3 *Labels* na cor branca com os nomes de cada coluna. Em seguida temos 5 arranjos iguais, cada um com também 3 *Labels* responsáveis por mostrar as informações de cada colocado, todos com fundo branco para dar um efeito listrado e ser visualmente mais fácil de observar. No fim da tela temos dois Botões, o primeiro volta para tela “Index” e o segundo é responsável por “zerar” os dados do *rank*, porém antes de executar a ação ele utiliza um *Notifier* para confirmar se o usuário deseja realmente fazer está ação. Ainda contamos com um *TinyDB* que oferece os dados dos colocados.

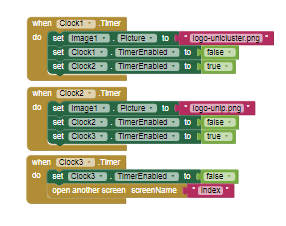


Créditos – Tela onde apresentamos os responsáveis pelo desenvolvimento deste jogo. Nela temos o logo do “Unicluster” e da “Unip” e também alguns *Labels* com informações dos desenvolvedores, coordenadores e versão. Por último temos um Botão voltar para tela “Index”

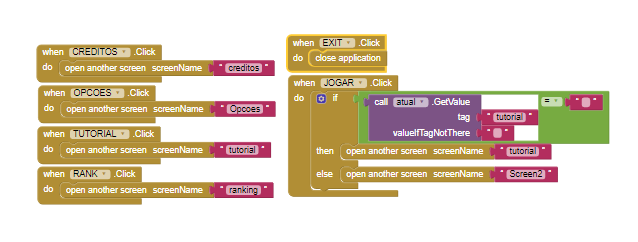


GameOver – Está é a tela que ninguém quer ver. Nela temos o cabeçalho padrão, com a diferença que o título é apresentado como uma Imagem. Temos dois arranjos com dois *Labels* iguais, neles temos as pontuações que o jogador obteve na partida, “Score” total de cores acertadas e a maior sequência que ele acertou. No fim da página temos 3 Botões, o primeiro volta para tela “Index” o do meio oferece a opção de salvar o seu “Score” no *rank,* e o último volta para tela do jogo “Screen2”. Caso o usuário clique em “Salvar Score” uma novo arranjo aparece na tela, com um campo para ele inserir seu nome e um Botão para confirmar, em seguida utilizamos os 2 *TinyDB* disponíveis para verificar a pontuação do jogador e dos 5 melhores já salvos, por fim o *Notifier* avisa o usuário se o “Score” dele foi alto o suficiente para ser salvo no *rank* ou não.

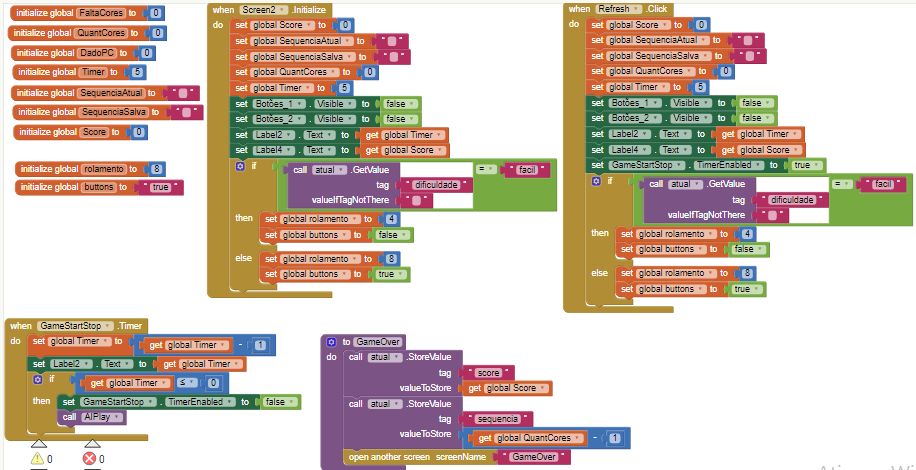
* 1. **Telas do Jogo (aba Bloco)**



Screen1 – Temos 3 blocos relacionados ao acionamento dos Relógios. O “Clock1” já vem ativado, quando ele é acionado a “Image1” muda, ele é desativado e o “Clock2” é ativado. O “Clock2” faz exatamente a mesma coisa que o 1, a diferença é que ele troca para outra imagem a “Image1” e desativa o “Clock2” e ativa o “Clock3”. Já o 3 quando é acionado apenas se desativa e abre uma nova tela a “Index”.

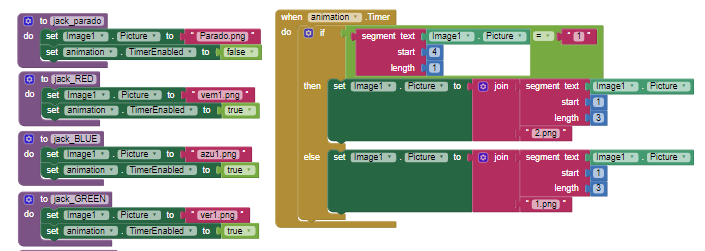


Index – Os 4 blocos da esquerda são praticamente iguais, eles são ativados ao se clicar em seus respectivos Botões, e cada um deles abre uma nova tela, referente ao Botão apertado. Na direita o primeiro bloco é ativado ao se clicar no Botão “EXIT” fechando a aplicação (esse botão é representado por um X no canto direito superior da maioria das telas, e por estar presente na maioria das telas e ser sempre igual, não será repetidamente apresentado futuramente neste documento). Por último temos o bloco ativado pelo Botão “JOGAR”, neste bloco temos um verificador de condição “IF” que verifica o valor encontrado na TAG “tutorial” utilizando o *TinyDB*, este processo visa verificar a opção definida pelo usuário no menu “OPÇÕES” relacionado ao tutorial, ou seja caso a opção esteja ativado (TAG = “ “) será exibido um tutorial antes de todas as vezes que ele for jogar, por tanto abrindo a tela “tutorial”, já se ela estiver desativada (TAG diferente de “ “) abre a tela do jogo direto “Screen2”.

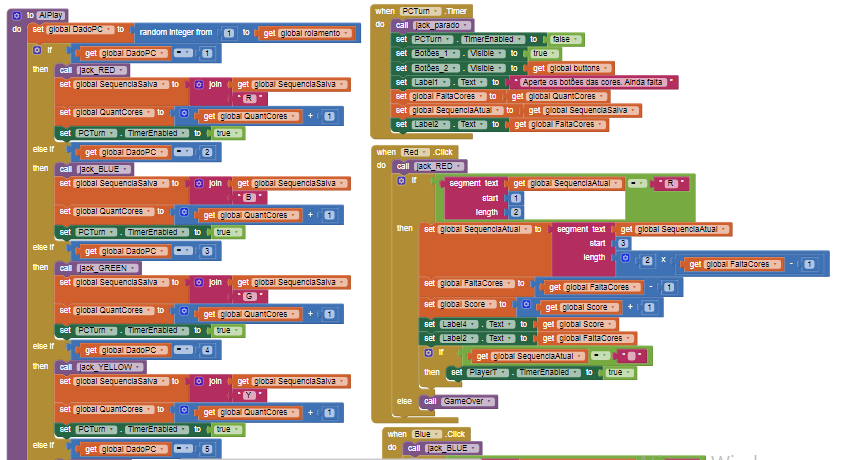


Screen2 (part1) – A tela responsável pelo jogo em si. Na esquerda superior temos diversas variáveis sendo iniciadas para seus valores padrões, de cima para baixo temos, FaltaCores que marca quantas cores falta apertar, QuantCores que marca quantas cores a sequencia tem no total, DadoPC que recebe um número variável responsável pela criação da sequência de cores variadas, Timer que marca o tempo para começar a partida, SequenciaAtual que utiliza para marcar quanto da sequência já foi completado, SequenciaSalva que recebe uma série de letrar em um formato padrão para marcar qual é a sequência de cores, Score que marca quantas cores o jogador já acertou, rolamento que recebe o valor responsável por fazer o jogo utilizar 4 cores (fácil) ou 8 (difícil), buttons que recebe o valor relacionado a visibilidade do segundo arranjo de botões que muda de acordo com a dificuldade.

A direita temos o bloco acionado quando a “Screen2” é inicializada, ele é responsável por deixar as algumas variáveis com seu valor padrão inicial, desabilitar a visibilidade do arranjo de botões, e por fim um verificador “IF” responsável por verificar no *TinyDB* qual a dificuldade selecionada no menu Opção e ajustar as variáveis rolamento e buttons de acordo com o resultado modificando a quantidade de cores utilizadas no jogo de 8 no difícil para 4 no fácil. Continuando a esquerda temos o bloco acionado quando o botão “Refresh” é acionado, este bloco funciona igual ao anterior, com apenas a adição de ativar o *timer* do Relógio “GameStartStop”. Em baixo a esquerda temos o bloco referente ao acionamento do *timer* do Relógio “GameStartStop”, que acontece a cada 1000ms, este bloco primeiro altera o valor da variável Timer diminuindo ele em 1 do seu valor original, então altera o valor do texto “Label2” para ser o valor atual da variável Timer e depois verifica se o valor da variável Timer é menor ou igual a zero, caso seja ele desabilita a si mesmo e em sequência chama o *Procedure* “AIPlay”. O último bloco é o *Procedure* “GameOver”, ele salva a variável Score no *TinyDB* e também a variável QuantCores (subtraindo 1, visto que se o jogador perdeu ele não acertou a quantidade de cores atual, logo a maior que ele acertou foi a anterior que é igual a atual menos um) e em seguida abre a tela “GameOver”.

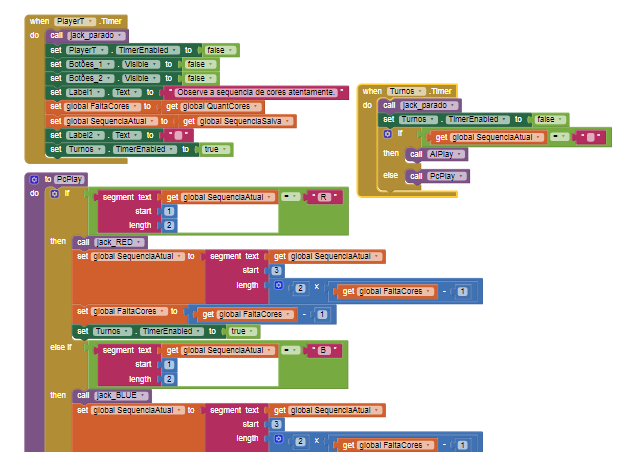


Screen2 (part2) – A esquerda temos alguns *Procedures* (são 9 no total, mas como 8 são praticamente iguais não havia necessidade de coloca-los na foto) responsáveis por fazer a Imagem principal da tela mudar, das 9 temos 1 que representa a tela em um estado inicial, o “jack\_parado” podemos ver que neste processo não temos a ativação do *timer* do Relógio “animation”, já nos outro 8 temos ativação do relógio, e mudamos a *Picture* da Imagem em cada um especificamente. Tirando o “jack\_parado” que tem apenas uma imagem parada, os outros processos são compostos por um par de imagens para dar um efeito de animação, este efeito é dado pelo bloco da direita que é acionado pelo *timer* do Relógio, então utiliza um verificador de condições “IF” para obter qual número temos no final do nome da imagem atual, este processo é possível ao se padronizar os nome das imagens, para cada processo demos um nome relacionado a uma cor, por exemplo o processo “jack\_RED” tem a cor vermelha, sendo assim o par de imagens é nomeado utilizando as 3 letras da cor, no caso “vem” , seguido pelo número da imagem “1” ou “2” e terminado pela extensão padrão “.png”, desta forma podemos utilizar o comando “*segment text”* para cortar o nome da imagem a partir do 4 caractere, pegando apenas 1 caractere, resultando no número 1 ou 2, independente do prefixo presente no nome relacionado da cor, seja ele “vem” ou “azu”, e em seguida trocar pelo outro número( se for “1” virá “2” e vice-versa) e remontar o nome utilizando o mesmo prefixo, assim mantemos o par correto.



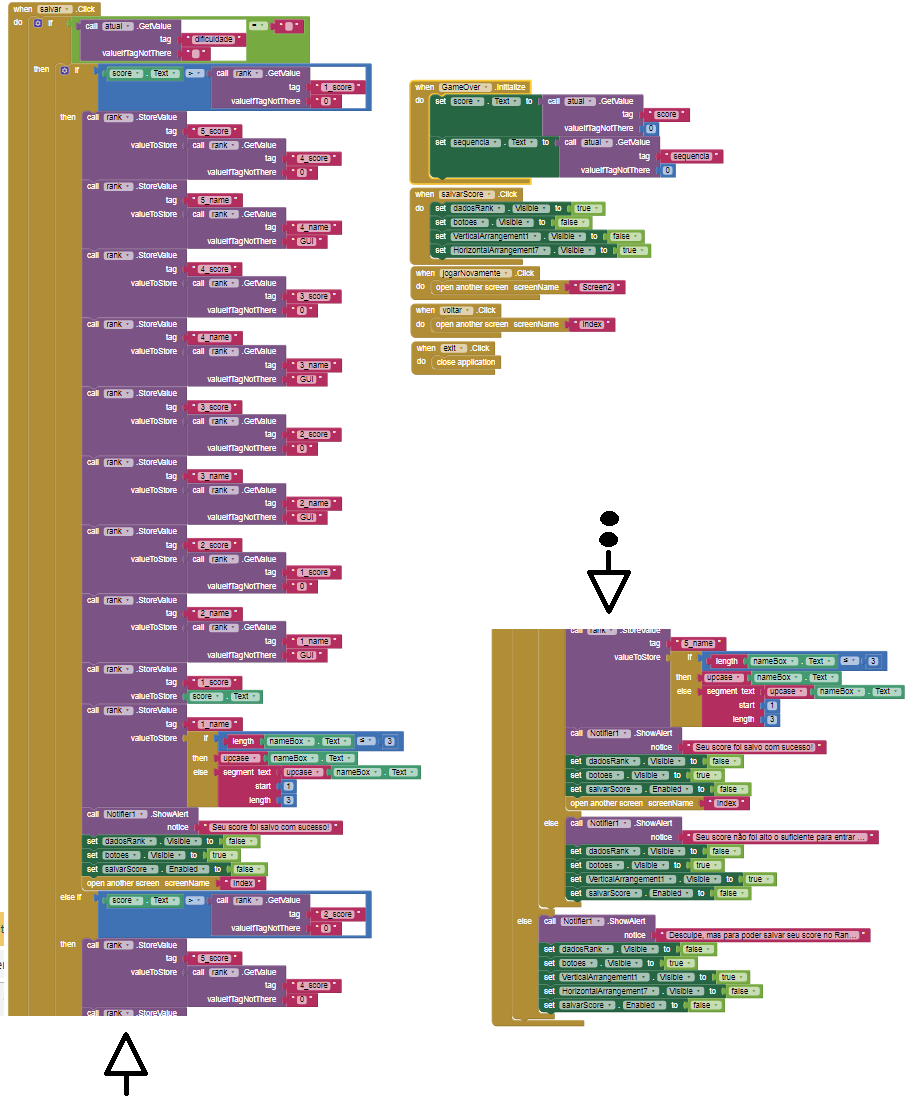
Screen2 part3 – A esquerda temos um bloco referente ao *Procedure* “AIPlay”, primeiro modificamos a variável DadoPC para um valor randômico entre 1 e a variável rolamento, que pode ser 4 ou 8 dependendo do nível de dificuldade, após nos temos um verificador de condições “IF” que executa uma função similar dependendo do número atribuído a variável DadoPC, para tanto atribuímos 8 verificações, uma para cada número, verificado o número o que é executado é bem similar. Primeiro chamamos um dos processos responsáveis pela troca da imagem principal, o processo chamado varia de acordo com o número definido a variável DadoPC, sendo um processo para cada número especificamente. Em seguida é atribuído a variável SequenciaSalva um valor formato pela junção do valor atual da própria SequenciaSalva com uma nova letra seguida de um espaço, por exemplo “B “, está letra pode variar dependendo do valor do DadoPC, para cada valor é designado uma letra diferente( 1-R; 2-B; 3-G; 4-Y; 5-P; 6-C; 7-M; 8-O ), mas sempre seguindo o mesmo padrão. Após é definido um novo valor para a variável QuantCores, este valor é dado pela soma do valor atual da variável com o numero 1, como o processo “AIPlay” é responsável por adicionar uma nova cor na sequência e a variável QuantCores marca a quantidade de cores que a sequencia tem, cada vez que adicionamos uma cor a sequência, também aumentamos o número marcado em 1. Por último é ativado o *timer* do Relógio “PCTurn”.

Na direita em cima temos o bloco relacionado ao *timer* do Relógio “PCTurn”, ele é responsável por chamar a animação padrão “jack\_parado”, desativar seu próprio *timer*, habilitar a visibilidade dos arranjos de Botões sendo o segundo somente habilitado caso o jogo esteja no difícil, para tanto utilizamos o valor encontrado na variável buttons definido em procedimento anterior, atribuir a variável FaltaCores o valor encontrado na variável QuantCores e também atribuir a SequenciaAtual o valor encontrado em SequnciaSalva e por fim mudar os textos do “Label1” e “Label2”. Na parte inferior da direita temos o bloco associado ao clique no Botão “Red” e mais abaixo podemos ver o começo do bloco do Botão “Blue”, apesar de não estar na imagem temos mais 6 blocos, um para cada botão que o jogador pode apertar, como são praticamente iguais não há necessidade de apresenta-los visualmente. Este bloco primeiro chama um dos processos de animação da imagem principal, referente a cor apertada, por exemplo se for apertado o botão “Red” será chamado o processo “jack\_RED”, já se apertar o Botão “Blue” será evocado o processo “jack\_BLUE” e assim por diante referente aos 8 botões. Em seguida temos um “IF” para verificar se a cor selecionada pelo jogador é a correta na ordem exigida, para tanto fazemos um processo de segmentação de texto utilizando a variável SequenciaAtual (por isso criamos está variável, assim podemos utiliza-la enquanto a variável SequenciaSalva permanece intacta com a sequencia completa) para retirar os 2 primeiros caracteres presentes nela, e verificamos se o que foi extraído é igual aos caracteres que representam o Botão apertado pelo jogador, por exemplo para o Botão “Red” temos o caractere “R “, para o “Blue” temos “B “, assim por diante para os 8 Botões. Se o jogador estiver certo o resultado obtido de segmentar a variável deve ser também “R “ visto que quando a sequencia é criada utilizamos o mesmo padrão de nomeação. Caso esteja errado evocamos o processo “GameOver”. Caso esteja certo o procedimento é atribuir um novo valor a variável SequenciaAtual, já que acertou a cor está precisa ser retirada da sequência de forma que na próxima verificação seja extraído a próxima cor da sequência ou que ela fique vazia quando for a última cor, para tanto utilizamos novamente o segmentador de textos no próprio valor da SequenciaAtual e segmentamos o texto a partir do 3 caractere, dessa forma excluímos os 2 primeiros que já foram utilizados na verificação anterior, e mandamos cortar até uma distancia de 2 vezes o valor da variável FaltaCores subtraído de um ( fazemos 2 vezes pois cada cor é inserida na sequencia utilizando 2 caracteres, multiplicamos pelo valor de FaltaCores pois está variável contem a quantidade de cores que ainda faltam acertar no momento e subtraímos 1 pois como o jogador acabou de acertar uma cor é necessário o ajuste). Em seguida ajustamos o valor da variável FaltaCores ao subtrair um e da variável Score ao adicionar 1. Após atualizamos os textos das *Label4 e Label2*. E por último utilizamos mais um “IF” para verificar se a sequencia ficou vazia, ou seja se a cor que foi acertada era a última que faltava, caso seja é ativado o *timer* do Relógio “PlayerT”, caso contrario não acontece nada.



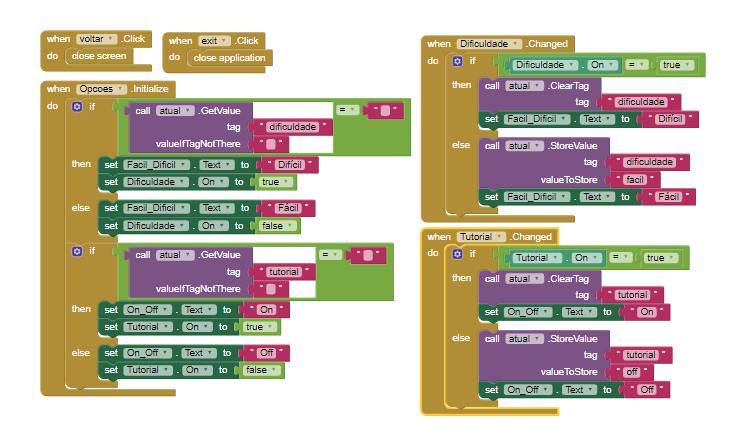
Screen2 part4 – Em cima do lado esquerdo temos o bloco associado ao *timer* do Relógio “PlayerT”, este bloco será acionado após o jogador fazer sua ultima jogada, ele prepara a interface para que o “Computador” possa jogar, para tanto ele primeiro chama o processo “jack\_parado” para transformar a imagem principal em sua forma inicial, em seguida ele se desabilita assim como torna a visibilidade dos botões utilizados pelo jogador como falsa (deixa invisível), modifica os textos em *Label1 e Label2*, atualiza o valor das variáveis FaltaCores e SequenciaAtual para respectivamente os valores encontrados em QuantCores e SequenciaSalva e por fim ativa o *timer* do Relógio “Turnos”.

A direita temos o bloco do Relógio “Turnos”, ele serve para dar um tempo entre cada cor apresentada pelo computador (as vezes o “computador” acaba por “jogar” 2 cores iguais na sequência, para não confundir o usuário foi criado este relógio, que faz a imagem principal voltar ao estado inicial entre cada cor, ou seja se a cor escolhida for vermelho por 2 vezes seguidas, aparecerá a animação referente, depois a animação inicial e então novamente a animação referente, caso o relógio não fosse utilizado seria a animação seria uma coisa só continua, e seria mais difícil para o usuário perceber que foi jogado vermelho por 2 vezes e não apenas uma) e também utiliza um bloco “IF” para verificar se a sequência já foi reproduzida até o fim, ou seja a variável SequênciaAtual está vazia, e então evoca o procedimento que faz o computador aleatoriamente inserir uma nova cor na sequência, já apresentado anteriormente o “AIPlay”, que ao fim vai dar ao jogador a possibilidade de jogar de novo, criando assim o *Loop* necessário para que o jogo dure “para sempre” até o jogador perder. Caso a sequência ainda não tenha sido completada, é evocado o processo “PCPlay”. Por último na esquerda em baixo temos o processo “PCPlay”, ele é composto por um grande bloco “IF” com um total de 8 “IF’s”, todos bem similares, para cada um utilizamos a verificação lógica comparando um trecho segmentado extraído da variavel SequenciaAtual, pegando os 2 primeiros caracteres, que de acordo com o padrão representam uma das cores da sequência, e verificamos, um a um, em cada um dos “IF”, qual cor está sendo representada pelo trecho ao compara-lo com os possíveis caracteres. Por exemplo, vamos supor que a próxima cor da sequência seja verde, quando extrairmos o trecho de texto da variável vamos obter “G “, esse trecho será comparado por cada um dos “IF” até encontrar o correto e então executará algumas ações, bem similares entre cada um dos “IF”. Primeiro será evocado um dos processos que muda a imagem principal, para cada um dos “IF” existe um processo especifico a ser chamada de acordo com a cor da sequência. Se como no exemplo temos a cor verde, o processo evocado será o “jack\_GREEN”, com imagens no tom verde. Os procedimentos seguintes são iguais em todos os “IF”, a variável SequenciaAtual tem seu valor alterado, visto que uma cor da sequencia foi utilizada, devemos retira-la da variável para que na próxima leitura seja executada a próxima cor), para tal segmentamos o texto da própria variavel em questão, pegando o trecho a partir do 3 caractere, visto que os 2 primeiros se referem a cor utilizada, e indo até caractere dado pelo dobro do valor na variavel FaltaCores subtraído de um. Em seguida atualizamos o valor da variavel FaltaCores para seu próprio valor subtraído de um, assim marcamos quantas cores faltam para a sequencia acabar. Por fim habilitamos o *timer* do Relógio “Turnos”.



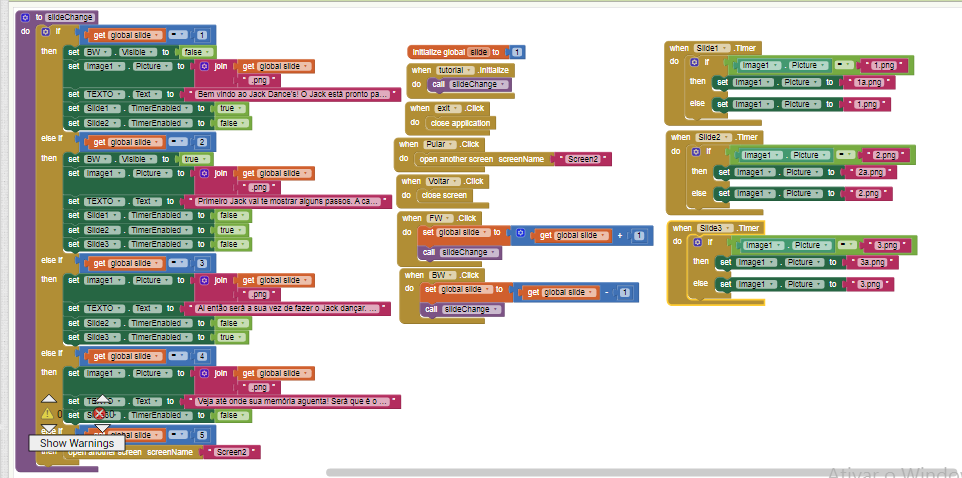
GameOver – Está tela que aparece automaticamente quando se perde o jogo. No canto superior a direita temos o bloco que é ativado quando a tela é inicializada, ele atualiza o texto das *Labels* “score” e “sequencia” pelos valores obtidos no *TinyDB* com as *TAGS* “score” e “sequencia” respectivamente. Em seguida temos o bloco acionado pelo clique no Botão “salvarScore”, ele deixa alguns arranjos invisíveis e outros novos visíveis, isso torna visível um novo campo para incluir as inicias que serão salvas, instruções sobre como faze-lo, e um botão para salvar definitivamente nomeado “salvar”. Abaixo temos 3 blocos associados ao apertar Botões, o primeiro “jogarNovamente” abre a tela “Screen2”, abaixo o bloco do botão “voltar” que abre a tela “Index” e por fim o do botão “exit” que fecha a aplicação.

Por último temos o bloco que se inicia na lateral esquerda da imagem, e tem a parte final no canto inferior direito, conforme as setas, sendo que parte intermediaria do conteúdo do bloco foi escondida por conter procedimentos repetitivos, que são iguais aos já contidos na imagem. Esse bloco é executado quando o usuário pressiona o botão “salvar”, ele primeiro utiliza um bloco “IF” para verificar em que dificuldade o jogo está, caso esteja no fácil será encontrado o valor “fácil” na *TAG* “dificuldade”, o teste lógico associado ao bloco “IF” irá falhar e será executado o processo que podemos ver no fim do bloco que primeiro utiliza o *Notifier* para enviar ao jogador uma mensagem explicando que apenas jogando na dificuldade difícil que ele poderá salvar seu *score* no *rank*, na sequencia os arranjos que tinham sido desabilitados previamente para o jogador poder incluir suas inicias serão novamente habilitados e o botão “salvarScore” será desabilitado para não executar sem necessidade o processo anterior. Caso o jogo esteja no difícil, o resultado encontrado na *TAG* será compatível com o teste lógico, e então iremos proceder utilizando um novo bloco “IF” com cinco verificações relacionadas ao *rank,* cada uma delas verifica se a pontuação atual é maior que, na exata ordem, do 1º, 2º, 3º, 4º e 5º colocados. Caso ela seja maior que algum dos que já estão classificados será executado um processo similar. Todo processo começa sobrescrevendo os dados salvos no *TinyDB* relativos ao 5º colocado, para tanto substituímos os dados que estão sob as *TAGS* “5\_score” e “5\_name” pelo que encontramos nas *TAGS* “4\_score” e “4\_name”, desta forma estamos basicamente passando quem estava em 4º lugar para o 5º lugar, seguiremos fazendo este processo, seguindo a ordem de trás para frente, ou seja do 5º seguindo para o 1º para que os dados não sejam perdidos, e por fim a diferença entre cada processo é onde iremos parar, por exemplo vamos supor que o jogador fez uma pontuação que supere o 3º colocado mas não o 2º, desta forma quando utilizarmos o segundo bloco de “IF”, ele irá falhar na primeira verificação, que verifica se é maior que o 1º colocado, depois irá falhar na segunda verificação, e na terceira vez ele irá passar, ai então o processo de sobrescrever os colocados se iniciará mas irá apenas até o passo no qual ela sobrescreve o 4ºcolocado com as informações do 3º, ai então ele irá sobrescrever as informações do 3º colocado com as informações do jogador encontradas na *Label* “score” e no *nameBox*, por fim irá exibir um aviso ao jogador explicando que o processo foi executado com sucesso e abrirá a tela “Index”. Caso a pontuação não seja maior que nenhum dos cinco colocados, o bloco “IF” irá executar o que pode ser visualizado no fim do bloco, no canto inferior direito da imagem, exibirá uma mensagem para o jogador informando que a pontuação não foi alta o suficiente, irá desabilitar os arranjos que foram previamente habilitados para o jogador inserir o seu nome, irá reabilitar os arranjos com os botões inicias, e por fim desabilitará o botão “salvarScore” para não executar o processo sem necessidade novamente.



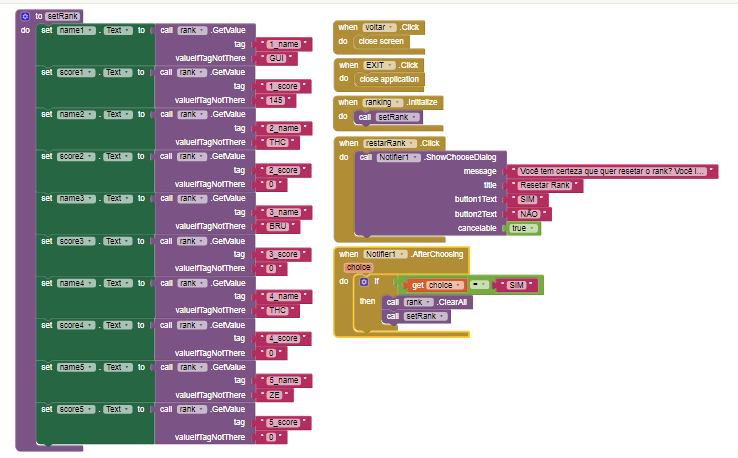
Opções – Os primeiros dois blocos da parte superior esquerda, são associados ao apertar Botões, o primeiro “voltar” fecha a tela atual, e o segundo “exit” fecha a aplicação. Abaixo temos o bloco executado quando a tela inicializa, ele utiliza dois blocos “IF” em sequencia para fazer duas ações iguais, mas em itens diferentes, o primeiro é executado na opção Dificuldade presente no menu desta tela, e o segundo na opção Tutorial. O bloco utiliza um comparador de igualdade para verificar qual o valor encontrado na *TAG* respectiva, de acordo com o resultado ele modifica o texto do respectivo *Label* e modifica o *Switch* referente para que fique de acordo com a opção, na posição de ligado ou desligado.

Com a mesma lógica do lado direito temos 2 blocos basicamente iguais que são executados quando o respectivo *switch* é utilizado. Estes blocos utilizam um “IF” para verificar em que está o *switch* se encontra, e então executar a ação especifica na qual ele atualiza o valor na *TAG* respectiva e muda o texto da *Label* referente para que o usuário visualize qual opção está definida no momento.



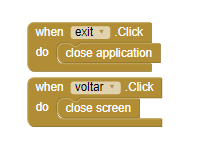
Tutorial – No centro temos a inicialização de uma variável “slide” que irá servir para marcar em qual dos passos do tutorial o usuário está. Abaixo quando a tela inicializar, chamaremos o processo “slideChange” para que ele defina a interface ao estado inicial, em seguida um “exit”, depois temos o bloco do Botão “Pular” que abre a tela “Screen2”, assim o usuário pode pular o tutorial, em seguida um bloco do Botão voltar que fecha a tela atual, o Botão “FW” que primeiro define o valor da variável “slide” para o seu valor atual mais 1 e em seguida chama o processo “slideChange”, e por último o bloco do Botão “BW” faz a mesma coisa que o bloco anterior, mas ao invés de adicionar ele subtrai 1, desta forma o usuário pode navegar entre os slides. Na esquerda temos o bloco do processo “slideChange”, ele começa utilizando um “IF” para fazer 5 verificações e descobri em que passo do tutorial estamos agora, encontrado o passo ele executa um processo parecido do passo 1 ao 4, primeiro ele deixa visível ou invisível Botão “BW” caso seja pertinente, depois ele troca a imagem principal da tela, os nomes das imagens foram selecionados de forma que ao se juntar o valor encontrado na variável “slide” mais a *string* “.pgn” teremos o nome da imagem referente ao slide em questão. Em seguida é alterado o texto da *Label* “TEXTO” para podermos ir explicando ao usuário como jogar, e por último são ativados e desativados os Relógios usados para animar a imagem nos casos pertinentes. Caso o valor encontrado na variável seja de 5 é então aberto a tela “Screen2”, levando o jogador para o jogo em si.

Por último na direita temos 3 blocos associados a Relógios, cada um para sua imagem especifica, para tanto quando o Relógio em questão dispara, utilizamos um “IF” que verifica o nome da imagem principal, depois de verificado o nome ele troca a imagem por outro e vice-versa, para dar um efeito de movimento a imagem.



Rank – A direita, de cima para baixo, temos o bloco do Botão “voltar” que fecha a tela atual, o bloco do Botão “exit”, o bloco executado quando a tela atual é inicializada que chama o processo “setRank”, o bloco do Botão “restarRank” que chama o *Notifier* para exibir uma mensagem pedindo que o usuário confirme que deseja apagar os dados salvos anteriormente no *rank*, por último o bloco que é executado após a escolha do usuário sobre apagar o *rank* para tanto ele utiliza uma variável que contem a resposta escolhida pelo usuário, e então um bloco “IF” que verifica se a resposta foi “SIM”, caso seja utilizamos o comando do *TinyDB* que paga os dados salvos nele e em seguida chamamos o processo “setRank” para atualizar os dados visuais do *rank*.

Do lado esquerdo temos o processo “setRank”, é um processo bem simples que busca pelos dados do *rank* como pontuação e nome de cada colocado, salvos no banco de dados, e então atualiza os textos das *Labels* referentes com estes dados.



Creditos – A tela Creditos apresenta apenas 2 blocos que já foram previamente explicados.

1. **Publicação**
   1. **Detalhes do APP**

**Título**: Jack Dance’s

**Breve descrição** (80 Caracteres): Venha testar sua memória e dançar com Jack.

**Descrição Completa** (4000 Caracteres):

Como está sua memória? Normalmente nos damos ao trabalho de exercitar nossos músculos, todos nós sabemos a importância que isso tem, mas e o seu cérebro? Apesar de não ser um músculo ele também precisa de exercícios constantemente.

Aqui está um jeito divertido de treinar sua memória e botar seu cérebro para funcionar. Venha ver as danças do Jack enquanto você pratica um jogo bem legal para exercitar os miolos.

Bata recordes, vá mais longe, entre para o Rank dos melhores jogadores. Será que você consegue?

**Código da versão:** 1.0

**O que há de novo na versão** (500 Caracteres):

Dois modos de jogo. Até 8 cores diferentes para desafiar sua memória a ir mais longe. Sistema de rank para os 5 melhores.

**Este app é**: (Gratuito)

**Países**: Brasil

**Contém anúncios**: (Não)

**Vídeo promocional (**Vídeo do YouTube)

**Insira um URL:** Clique ou toque aqui para inserir o texto.

**E-mail de contato**: guiga.benitez@gmail.com

* 1. **Recursos Gráficos**

|  |  |
| --- | --- |
| **Ícone de alta resolução**  512 (largura) x 512 (altura) PNG de 32 bits (sem alfa) | **Gráfico promocional**  180 (largura) x 120 (altura) JPG ou PNG de 24 bits (sem alfa) |

|  |  |
| --- | --- |
| **Capturas de tela**  (Tela Inicial, Tutorial, Ranking, 4 Telas das Fases e Créditos)  JPEG ou PNG de 24 bits (sem alfa). Mínimo 320px e máximo 3840px de largura, com proporção de escala em 2:1. | |
|  |  |
|  |  |
|  |  |
|  |  |

* 1. **Classificação de Conteúdo**

**Violência:**

O jogo contém inferências, referências ou representações de violência?

O jogo contém inferências, referências ou representações de violência?

|  |  |
| --- | --- |
| Sim | Não |

**Medo:**

O jogo contém inferências, referências ou representações de violência?

O jogo contém imagens ou sons assustadores ou aterrorizantes?

|  |  |
| --- | --- |
| Sim | Não |

**Sexualidade:**

O jogo contém inferências, referências ou representações sexuais, violência sexual, insinuação, trajes sedutores ou nudez?

|  |  |
| --- | --- |
| Sim | Não |

**Apostas Simuladas, Reais ou pagamentos em Dinheiro:**

O jogo contém apostas, simulações de jogos de cassino/bingo ou pagamentos em dinheiro reais?

|  |  |
| --- | --- |
| Sim | Não |

**Linguagem**:

O jogo contém linguagem potencialmente ofensiva?

|  |  |
| --- | --- |
| Sim | Não |

**Substâncias Controladas**:

O jogo contém referência ou uso de drogas, álcool ou tabaco?

|  |  |
| --- | --- |
| Sim | Não |

**Humor Grosseiro:**

O jogo contém funções do corpo como arroto, flatulência ou vômito quando usadas para fins humorísticos?

|  |  |
| --- | --- |
| Sim | Não |

**Diversos**:

O jogo permite nativamente que os usuários interajam ou troquem conteúdo com outros usuários por meio de comunicações por voz, texto ou compartilhamento de imagens ou áudio?

|  |  |
| --- | --- |
| Sim | Não |

O jogo compartilha a localização física atual do usuário com outros usuários?

|  |  |
| --- | --- |
| Sim | Não |

O jogo permite aos utilizadores comprar bens digitais?

|  |  |
| --- | --- |
| Sim | Não |

O jogo contém suásticas ou outros símbolos semelhantes e/ou propaganda nazistas que são considerados inconstitucionais na Alemanha?

|  |  |
| --- | --- |
| Sim | Não |

O jogo contém conteúdo que pode substancialmente desgastar a identidade nacional da República da Coréia ao descrever atos antinacionalistas ou fatos históricos distorcidos? \*

|  |  |
| --- | --- |
| Sim | Não |

O jogo contém descrições detalhadas de técnicas que podem ser utilizadas ​​em crimes?

|  |  |
| --- | --- |
| Sim | Não |

O jogo defende atos de terrorismo?

|  |  |
| --- | --- |
| Sim | Não |

**Detalhes da Classificação**:

Clique ou toque aqui para inserir o texto.

**Requisitos de Entrega**

**Objetivo:**

Criação de um jogo com as seguintes características:

**Splash Screen:**  Logo Unicluster e da Unip por 2 segundos e então ir para a tela inicial.

**Tela Inicial com os botões:**(manter a sequência)

* Jogar
  + Pelo menos 2 fases
    - Dificuldade e Cenários Distintos.
* Tutorial
  + Demonstra as ações que o personagem possui no jogo.
* Ranking
  + Guardar o último, mas de preferência os 3 últimos resultados.
* Crédito
  + Seguir o padrão do moodle.
* Sair

**Como entregar:**

A atividade deve ser em dupla (**não pode ser a mesma dupla NP1**) e entregue no moodle por apenas um dos alunos até **05/06** as 23:59.

* arquivo **.docx** (este documento)
* arquivo **.apk**
* arquivo **.aia**
* Imagens para publicação (**.zip**)
  + Ícone: **512 x 512 - PNG de 32 bits - sem canal alpha**
  + Telas de 1 a 8 numeradas na ordem: **JPEG ou PNG de 24 bits (sem alfa). Comprimento mínimo de cada lado: 320 px. Comprimento máximo de cada lado: 3840 px. Proporção scala 2:1**
  + Tela Inicial, Tutorial, 4 telas das diferentes fases do Jogo, Ranking e Crédito
* Game Play (vídeo direto da tela do Celular)
  + vídeo de 2:00 (mp4)
  + inicia no splash screem, começa o jogo e vai até onde der 2:00
  + tem que ser uma sequência inteira e não pode perder

**Como apresentar:**

Todos os alunos devem comparecer no dia **06/06** para apresentação do jogo em um celular android e assinar a lista de presença.

Entregar uma via impressa APENAS DA CAPA por aluno.