Projeto 2 - Virtual Pet

UTFPR/DACOM 2019

BCC35A-Linguagens de Programação

Características gerais

- Objetivo: criar app com **Orientação a Objetos**
- Tipo: app desktop
  - Kit GUI
- Linguagens Client Side
  1. Ruby
  2. Python
  3. Dart
  4. Lua (LÖVE ou kit GUI)
  5. Type Script (Electron)
  6. Perl
- Será necessário utilizar libs/SDKs/frameworks de GUI/Draw 2D

Projeto: Virtual Pet

Virtual Pet

- Objetivo do App:
  - Imitar Tamagotchi original da Bandai
- Requisitos gerais:
  - **Orientação a Objetos**
  - Motor do VPET
  - UI: botões, barras, etc...
  - Gráficos: pet, ícones, indicadores, etc...
  - Sons
  - Storage: arquivos/BD/Não-relacional

Virtual Pet

- Criar App de **Virtual Pet**
  - Tamagotchi (Bandai)
  - https://en.wikipedia.org/wiki/Tamagotchi
- App Similar: Hatchi
  - https://play.google.com/store/apps/details?id=com.portablepixels.ht
- Manual do Tamagotchi
  - http://www.mimitchi.com/tamaplus/manual.shtml

Avaliação e Questões Gerais

- **Cópias**: Qualquer tipo de cópia (trabalhos de colegas, internet, etc) anulará o trabalho
  - Seja por porções de código ou pelo trabalho completo
- **Entrega**: código fonte pelo Moodle
- **Data**: informada na página do Moodle
- **Ferramentas**: IDE/editor e framework/SDK
- **Critérios**:
  - Mínimo: implementar app -> nota mínima
  - Diferencial: qualidade e recursos adicionais -> incrementa até a nota máxima

Projeto do VPet: controle

Características gerais

- Controle de usuários/criadores
  - Login/senha ou mais simples (somente login)
  - Cada usuário pode ter varios PETs
- Ações gerais
  - Criar VPet
  - Remover VPet
- Ranking e gráficos comparativos de criadores
  - Tempo de vida de cada pet (para todos os criadores)
  - Gráficos gerais (Happy, Hunger, Health, estatíticas de minigames...)
- Novo Pet
  - Informar nome

Atributos / Status (versão básica)

- Barras de status (0..100)
  - **Happy**: felicidade
    - ★ Diminui com o tempo
  - **Hunger**: fome
    - ★ Diminui com o tempo
    - ★ Diminui ao brincar
  - **Health**: vitalidade do pet
    - ★ Pet pode ficar doente por acúmulo de sujeira (tempo) ou estado **Sick**
    - ★ Diminui com o tempo; quando come sem necessidade (Feed)
  - Sugestões de mudanças de estados
    - ★ Se Health baixo -> pet fica Doente
    - ★ Se Helath zero -> pet morre (precisa reiniciar)

Estados do Pet

- Estados
  - Indicados por ícone ou desenho do personagem
- Estados básicos -> ações que os resolvem
  - **Normal**
  - **Sick** (doente) -> precisa de cura (Cure)
  - **Tired** (cansado) -> precisa descançar (Lights)
  - **Dirty/Toilet** (sujo/banheiro) -> precisa limpar/banheiro (Toilet/Flush)
  - **Sad** (triste) -> precisa brincar (Play)
  - **Sleeping** (dormindo) -> pode ser acordado a qualquer momento. Dormir recarrega Health e consome pouco Hunger (gradativamente).
  - **Dead** (morto) -> recomeçar pet

Ações

- **Feed**:
  - Aumenta **Hunger**
  - Em excesso: aumenta peso ou fica **Doente**
- **Toilet/Flush**:
  - Aumenta **Health** e elimina **Sujo/Banheiro**
  - Quando **Sujo/Banheiro**
- **Play**:
  - Aumenta **Happy**
  - Diminui **Hunger**
  - Minigames:
    - ★ Dance (jogo da memória)
    - ★ Jump (pular sincronizadamente)
    - ★ **Prezem pelo minigame**
    - ★ Pontos no minigame são traduzidos em pontos de **Happy**

Ações

- **Cure**:
  - Quando pet está **Doente**, use o comando para curá-lo com uma injeção. Pode ser necessário mais de uma dose. Por outro lado, curá-lo em estado normal pode levá-lo ao estado doente.
- **Lights**:
  - Quando o Pet estiver cansado (Ícone ou desenho do personagem) é necessário colocá-lo para domir. Para tando, use o comando para desligar as luzes. Caso não desligue, ele não descansará.

Controle

- Utilize taxas de aleatoriedade em ações e no incremento/decremento dos itens de status
  - Fica mais simples trabalhar com itens de status variando de 0 a 100 pontos (facil visualizar como percentual)
    - ★ Happy, Hunger e Health
  - Taxas de aleatoriedade podem ser trabalhas como percentuais
    - ★ randomness: pode ir de 0.8 a 1.2, por exemplo
    - ★ hunger -= (hungerRate * rand(0.8, 1.2)) * deltaTime
  - Implementar **máquina de estados** para controlar estados do vpet

App: Hatchi

Figure 1: Hatchi, Virtual Pet (Android/iOS)

Projeto do VPet: Lógica

“Motor” do VPet: função update(deltaTime)

- Criar função que atualiza a lógica do vpet
  - função update(deltaTime)
  - deltaTime: tempo desde a última atualização
    - ★ quando app aberto, chama update() por evento temporizado fixo e deltaTime terá um valor fixo
    - ★ quando app é fechado, persiste tempo atual
    - ★ quanto app é reabert, carrega tempo anterior, calcula diferença (deltaTime) e a usa na função update()
- Função update(deltaTime): “motor” do VPet
  - controla **máquina de estados** do vpet
  - atualiza os itens de status: Happy, Hunger e Health
  - define estado: Normal, Sick, Tired ou Dirty/Toliet
  - ao final, atualiza estado visual
    - ★ opcionalmente, persiste estado ao final da função update(dt)

“Motor” do VPet: função update(deltaTime)

- Itens de status devem ser consumidos pela função update(), considerando:
- As taxas de cosumo de cada item/barra de statis devem ser definidas de acordo com cada estado
  - estado **normal**, **healthRate** = 0.1
  - estado **doente**, **healthRate** = 0.3 (consome vitalidade 3x quando está doente)
- Exemplo de consumo dos itens/barras de status

```
// aleatoriedade nas taxas de decremento
// também podem estar relaciondas ao estado atual do pet
hunger -= (hungerRate * rand(0.8, 1.2))  * deltaTime
health -= (healthRate * rand(0.9, 1.1))  * deltaTime
happy  -= (happyRate  * rand(0.85, 1.15))* deltaTime
```

```
class VPet { // VERSÃO 1: carrega/grava estado ao iniciar/fechar app
    ...
    update(lastTime) {
        deltaTime = Time.currentTime - lastTime // delta desde último update()
        // máquina de estados do vpet
        if (state == 'normal') {
            // taxas estão relacionadas ao estado atual do Pet
            hungerRate = 5;  healthRate = 4;  happyRate = 3

            // atualiza itens de status (versão "muito simples")
            hunger -= (hungerRate * randomBetween(0.8, 1.2)) * deltaTime
            health -= (healthRate * randomBetween(0.9, 1.1)) * deltaTime
            happy  -= (happyRate * randomBetween(0.85, 1.15)) * deltaTime

            // atualiza estados
            if (this.happy < 25)     state = 'sad'
            elif (this.health < 25)  state = 'sick'
            elif (this.happy <= 0 || this.health <= 0 || this.hunger <= 0)
                this.state = 'dead'
        }
        else ...  // if (state == 'sick') ... outros estados

        // atualiza desenho do pet e ícones na tela
        this.updateGraphics()
    }
}
```

```
class VPet { // VERSÃO 2: persiste etado a cada update
    ...
    update() {
        deltaTime = Time.currentTime - Persistence.loadState().lastTime //***
        // máquina de estados do vpet
        if (state == 'normal') {
            // taxas estão relacionadas ao estado atual do Pet
            hungerRate = 5;  healthRate = 4;  happyRate = 3

            // atualiza itens de status (versão "muito simples")
            hunger -= (hungerRate * randomBetween(0.8, 1.2)) * deltaTime
            health -= (healthRate * randomBetween(0.9, 1.1)) * deltaTime
            happy  -= (happyRate * randomBetween(0.85, 1.15)) * deltaTime

            // atualiza estados
            if (this.happy < 25)     state = 'sad'
            elif (this.health < 25)  state = 'sick'
            elif (this.happy <= 0 || this.health <= 0 || this.hunger <= 0)
                this.state = 'dead'
        }
        else ...  // if (state == 'sick') ... outros estados

        // atualiza desenho do pet e ícones na tela
        this.updateGraphics()
        // salva estado: barras de status (number), estado (string) e hora (number)
        Persistence.saveState(hunger, health, happy, state, Time.currentTime)//***
    }
```