Projeto Tamagotchi / VPet

UTFPR/DACOM 2020-2

BCC35A-Linguagens de Programação

Projeto: Tamagotchi / VPet

- Criar App de **Virtual Pet** (Tamagotchi)
  - Simulador simples de virtual pet
- Linguagens
  - Imperativas Interpretadas e Compiladas
- Foco:
  - Imitar Tamagotchi original da Bandai
- Grupo: 2 alunos ou individual
- Inpiração
  - Tamagotchi (Bandai)
  - https://en.wikipedia.org/wiki/Tamagotchi

App: Hatchi

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

Formato e Requisitos

- Cliente/Servidor
  - Cliente:
    - ★ App VPet, desktop, UI gráfica
    - ★ IU: Sprites, sons, componentes UI
    - ★ Comunicação com servidor: requisições HTTP
  - Servidor:
    - ★ Responde à requisições HTTP
    - ★ Intercâmbio de dados em formato JSON
    - ★ Storage (server): relacional ou não (key-value)
- App Similares:
  - Hatchi (RIP)
  - Wildagotchi: Virtual Pet
  - Pou
  - My Chu 2
  - Duddu
- Manual do Tamagotchi
  - http://www.mimitchi.com/tamaplus/manual.shtml

Linguagens e tecnologias

- App VPet (cliente): emprego de SDK
  1. Java: JavaFX
  2. TypeScript: Electron
  3. C#: Godot, Xamarin
  4. Lua: LÖVE, Corona SDK, Gideros
  5. Python: Kivy, PyGUI
  6. Haxe: OpenFL, HaxeFlixel
- Serviço (servidor): HTTP, persistência
  1. PHP
  2. TypeScript
  3. Python
  4. Ruby
  5. C# (.Net) (Windows)

Visão Geral

- Orientação a Objetos + SDK
- App Cliente VPet
  - Acesso via usuário e interação geral
  - UI: botões, barras, etc. . .
  - Gráficos e sons: pet, ícones, indicadores, etc. . .
  - Motor do VPet (simulação)
  - Envio de "backup" para servidor (sincronização)
- Serviço HTTP
  - Um servidor <> múltiplos clientes
  - Storage: Banco de Dados (relacional ou não)
  - Seviços HTTP para apps clientes
    - ★ Sincronização (backup) cliente => servidor
    - ★ Recuperação de VPets por usuário

Avaliação e Questões Gerais

- **Cópias**: Qualquer tipo de cópia de código-fonte (trabalhos de colegas, internet, etc) anulará o trabalho
  - Seja por porções de código ou pelo trabalho completo
- **Entrega**: código fonte pelo Moodle
- **Apresentação**: síncrona, por sessão agendada
  - Serão feitas (várias) perguntas sobre o código e funcionalidades
- **Data**: informada na página do Moodle
- **Critérios**:
  - Requisitos, acabamento, diferenciais

Projeto do VPet: controle

Requisitos Gerais: Cliente

- Gerenciar usuários/criadores
  - Login/senha ou mais simples (somente login)
  - Cada criador pode ter varios PETs
- Gerenciar VPets
  - Criar, Listar, Alterar e Remover VPets
  - Ao criar VPet, pode escolher nome, visual, ...
- Cuidar de VPet
  - Verificar atributos
  - Observar estado
  - Realizar ações
  - Executar repostas visuais e auditivas (sprites, sons e UI)

Requisitos Gerais: Cliente

- Simular Vpet
  - Processamento local com envio de cópias dos dados ao servidor
  - Envio a cada atualização do VPet (deltaTime)
    - ★ Todo o controle é local: servidor apenas serve como storage remoto para backup e posterior restauração (em caso de instalação em novo dispotivo)
- Tratar erros/exceções e informar ao usuário
- **Opcional**:
  - Ranking de criadores
    - ★ Por tempo de vida dos VPets
    - ★ Outros critérios...

Requisitos Gerais: Servidor

- Dados a Persistir
  - Criadores
    - ★ usuario
    - ★ senha
    - ★ (outros que sejam necessarios. . .)
  - VPets
    - ★ nome
    - ★ tempo de vida
    - ★ última atualização
    - ★ tipo visual (usuáro pode escolher sprites)
    - ★ estado
    - ★ atributos
    - ★ dormindo
    - ★ (outros que sejam necessarios. . .)
- Relacionamento
  - Criador 1:N VPets

Operação do VPet: Atributos (S), Estados (E) e Ações (A)

Figure 2: Hatchi, Virtual Pet (Android/iOS)

Atributos / Status (versão básica)

- Atributos/Barras de status (0..100)
  - **S_Happy**: felicidade
    - ★ Diminui com o tempo
  - **S_Hunger**: fome
    - ★ Diminui com o tempo
    - ★ Diminui ao brincar (ação **A_Play**)
  - **S_Health**: vitalidade do pet
    - ★ Pet pode ficar doente por acúmulo de sujeira (por tempo) quando no estado **E_Dirty**, lenvando ao estado **E_Sick**
    - ★ Diminui com o tempo; quando come sem necessidade (**A_Feed**)
  - Sugestões de mudanças de estados
    - ★ Se **S_Health** baixo -> pet fica Doente
    - ★ Se **S_Health** zero -> pet morre
- **OBS**: podem ser acrescentados novos atributos

Estados do Pet

- Estados: indicados por ícones ou sprite do personagem
- Estados básicos -> ações que os resolvem
  - **E_Normal**
  - **E_Sick** (doente)
    - ★ Precisa de cura (**A_Cure**)
  - **E_Tired** (cansado)
    - ★ Precisa descançar (**A_Lights**)
  - **E_Dirty** (sujo/banheiro)
    - ★ Precisa limpar/banheiro (**A_Toilet**)
  - **E_Sad** (triste)
    - ★ Precisa brincar (**A_Play**)
  - **E_Sleeping** (dormindo)
    - ★ Pode ser acordado a qualquer momento. Dormir recarrega **S_Health** e consome pouco **S_Hunger** (gradativamente).
  - **E_Dead** (morto) Recomeçar/encerrar pet

Ações

- **A_Feed**:
  - Aumenta **S_Hunger**
  - Em excesso: aumenta peso ou fica **E_Sick**
- **A_Toilet**:
  - Aumenta **S_Health**
  - Elimina **E_Dirty** (necessário)
- **A_Play**:
  - Aumenta **Happy**
  - Diminui **Hunger**
  - Pode ser uma ação; **Opcional**: Minigame
    - ★ Dance (jogo da memória)
    - ★ Jump (pular sincronizadamente)
    - ★ Pontos no minigame podem ser traduzidos em pontos de **S_Happy**

Ações

- **A_Cure**:
  - Quando pet está **E_Sick**, use o comando para curá-lo com medicamento. Pode ser necessário mais de uma dose. Por outro lado, curá-lo em estado **E_Normal** pode levá-lo ao estado **E_Sick**.
- **A_Lights**:
  - Quando o Pet estiver cansado (indicado por 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.
- **OBS**: regras adicionais podem ser implementads sobre consumos de atributos, trocas de estados e efeitos de ações.

Implementação do Motor do VPet

Motor do VPet: função update(deltaTime)

- Criar método que atualiza a lógica do vpet
  - **update()**
  - **deltaTime**: tempo desde a última atualização
    - ★ quando app está aberto: chama update() por evento temporizado fixo (deltaTime tem valor fixo)
    - ★ quando app é fechado: persiste tempo atual
    - ★ quanto app é reaberto: carrega tempo anterior, calcula diferença (deltaTime) e a usa na função update()
- Função update(): motor do VPet
  - Controla **máquina de estados** do vpet
  - Atualiza os atributos: **S_Happy**, **S_Hunger** e **S_Health**
  - define estado: **E_Normal**, **E_Sick**, **E_Tired**...
  - Ao final, atualiza estado visual
    - ★ Persiste estado local e no servidor ao final da função update()

Motor do VPet: função update(deltaTime)

- Atributos (0..100) devem ser consumidos pela função update()
- As taxas de consumo de cada atributo devem ser definidas de acordo com o estado
  - estado **E_Normal**, **healthRate** = 1
  - estado **E_Sick**, **healthRate** = 3 (consome vitalidade 3x quando está doente)
- Exemplo de consumo dos atributos

```
// taxa de consumo * variação
// relacionda ao estado atual do pet
if (state == 'E_Normal') {
hunger -= (hungerRate * rand(0.8, 1.2))  * deltaTime
health -= (healthRate * rand(0.9, 1.1))  * deltaTime
happy  -= (happyRate  * rand(0.85, 1.15))* deltaTime
}
```

- Implementar **máquina de estados** para controlar estados do vpet

```
class VPet { ...
    update() {
        deltaTime = Time.currentTime - Persistence.loadState().lastTime
        // máquina de estados do vpet
        if (state == 'E_Normal') {
            // taxas estão relacionadas ao estado atual do Pet
            hungerRate = 5;  healthRate = 4;  happyRate = 3

            // atualiza atributos
            hunger -= (hungerRate * rand(0.8, 1.2)) * deltaTime
            health -= (healthRate * rand(0.9, 1.1)) * deltaTime
            happy  -= (happyRate * rand(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: atributos, estado, tempo, ...
        Persistence.saveState(hunger, health, happy, state, Time.currentTime, ...)
        Server.saveState(hunger, health, happy, state, Time.currentTime, ...)
    }
```