Component Items Module

Template Module

Item

Uses

ScriptableObject, Inventory, UnityEngine

Syntax

Exported Types

Sprite

Exported Access Programs

Routine name	In	Out	Exceptions
Use			
RemoveFromInventory			

Semantics

State Variables

name: *String*
icon: *Sprite*

State Invariant

None

Assumptions

This module is used to create a new asset in Unity by creating a new asset menu called "Inventory/Item". Once an item has been created through Unity's asset menu, the state variables are updated directly in Unity by manually typing in the name of the item, and placing the appropriate Sprite for the icon. User will not be tasked to do this; all items available to the user will be created before hand by the developers of Zombie Survival Kit.

Access Routine Semantics

Use():

- translation: None
- output: Prints out to Debug.Log()
- exception: None

RemoveFromInventory():

- translation: Calls the "Remove" method from the Inventory class.
- output: None
- exception: None

Component Items Consumable Module

Template Module

Consumable

Uses

Item, PlayerStats, System.Collections, System.Collections.Generic, UnityEngine

Syntax

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
Use			

Semantics

State Variables

healthModifier: $\mathbb{N}$
name: *String*
icon: *Sprite*

State Invariant

None

Assumptions

This module is used to create a new asset in Unity by creating a new asset menu called "Inventory/Consumable"; item of type Consumable. Once a Consumable has been created through Unity's asset menu, the state variables are updated directly in Unity by manually typing the value of the healthModifier, the name of the item, and placing the appropriate Sprite for the icon. User will not be tasked to do this; all Consumable items available to the user will be created before hand by the developers of Zombie Survival Kit.

Access Routine Semantics

Use():

- translation: Calls the base "Use" method from the Item class, the "Eat" method in the PlayerStats class, and "RemoveFromInventory" method from the Item class.
- output: None
- exception: None

Component Item Equipment Module

Template Module

EquipmentItem

Uses

Item, EquipmentManager, System.Collections, System.Collections.Generic, UnityEngine

Syntax

Exported Types

equipmentSlot: {Head, Chest, Legs, Primaryhand, Offhand, Feet}

Exported Access Programs

Routine name	In	Out	Exceptions
Use			

Semantics

State Variables

equipSlot: *equipmentSlot*
attackModifier: $\mathbb{N}$
defenceModifier: $\mathbb{N}$
name: *String*
icon: *Sprite*

State Invariant

None

Assumptions

This module is used to create a new asset in Unity by creating a new asset menu called "Inventory/Equipment"; item of type EquipmentItem. Once an EquipmentItem has been created through Unity's asset menu, the state variables are updated directly in Unity by manually choosing which equipmentSlot belongs to the EquipmentItem's equipSlot, typing in the value of the attackModifier, defenceModifer and the name of the EquipmentItem,

and placing the appropriate Sprite for the icon. User will not be tasked to do this; all EquipmentItem available to the user will be created before hand by the developers of Zombie Survival Kit.

Access Routine Semantics

Use():

- transition: Calls the base "Use" method from the Item class, the "Equip" method in the EquipmentManager class, and "RemoveFromInventory" method from the Item class.
- output: None
- exception: None

Generic Seq2D Module

Generic Template Module

Seq2D(T)

Uses

N/A

Syntax

Exported Types

Seq2D(T) = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
Seq2D	seq of (seq of T), $\mathbb{R}$	Seq2D	invalid_argument
set	PointT, T		outside_bounds
get	PointT	T	outside_bounds
getNumRow		$\mathbb{N}$	
getNumCol		$\mathbb{N}$	
getScale		$\mathbb{R}$	
count	T	$\mathbb{N}$	
count	LineT, T	$\mathbb{N}$	invalid_argument
count	PathT, T	$\mathbb{N}$	invalid_argument
length	PathT	$\mathbb{R}$	invalid_argument
connected	PointT, PointT	$\mathbb{B}$	invalid_argument

Semantics

State Variables

s: seq of (seq of T)
scale: $\mathbb{R}$

nRow: $\mathbb{N}$
nCol: $\mathbb{N}$

State Invariant

None

Assumptions

- The Seq2D(T) constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once.
- Assume that the input to the constructor is a sequence of rows, where each row is a sequence of elements of type T. The number of columns (number of elements) in each row is assumed to be equal. That is each row of the grid has the same number of entries. $s[i][j]$ means the ith row and the jth column. The 0th row is at the bottom of the map and the 0th column is at the leftmost side of the map.

Access Routine Semantics

Seq2D(S, scl):

- transition: [s, scale, nRow, nCol := S, scl, S.len, S[0].len —SS]
- output: $out := self$
- exception: $[scl < 0 \| \neg(validRow) \| \neg(validCol) \| S.len \neq S[0].len \implies invalid\_argument$ —SS]

set(p, v):

- transition: [s[p.y][p.x] := v —SS]
- exception: $[p.y < 0 \| p.x < 0 \| p.y \geq nRow \| p.x \geq nCol \implies outside\_bounds$ —SS]

get(p):

- output: [out:= s[p.y][p.x] —SS]
- exception: $[p.y < 0 \| p.x < 0 \| p.y \geq nRow \| p.x \geq nCol \implies outside\_bounds$ —SS]

getNumRow():

- output: $out :=$ nRow

- exception: None

getNumCol():

- output: out := nCol
- exception: None

getScale():

- output: out := scale
- exception: None

count(t: T):

- output: $[out := +(\forall i : \mathbb{N} | i \in [0...nRow] \bullet \forall j : \mathbb{N} | j \in [0...nCol] \bullet s[i][j] = t : 1)$ —SS]
- exception: None

count(l: LineT, t: T):

- output: $[out := +(\forall i : \mathbb{N} | i \in [l.strt.y...l.end.y] \bullet \forall j : \mathbb{N} | j \in [l.strt.x...l.end.x] \bullet s[i][j] = t : 1$ —SS]
- exception: $[\neg(validLine(l)) \implies invalid\_argument$ —SS]

count(pth: PathT, t: T):

- output: $[out := +(\forall k : \mathbb{N} | k \in [0...pth.size-1] \bullet \forall i : \mathbb{N} | i \in [pth[k].strt.y...pth[k].end.y] \bullet \forall j : \mathbb{N} | j \in [pth[k].strt.x...pth[k].end.x] \bullet s[i][j] = t : 1$ —SS]
- exception: $[\neg(validPath(pth)) \implies invalid\_argument$ —SS]

length(pth: PathT):

- output: $[out := scl \times pth.len$ —SS]
- exception: $[\neg(validPath(pth)) \implies invalid\_argument$ —SS]

connected(p_1: PointT, p_2: PointT):

- output: $[out := ((\forall k : \mathbb{N} | k \in [0...pth.size-1] | \bullet \exists p_1 : PointT | \bullet pointsInPath(pth)[k] = p_1) \wedge (\forall k : \mathbb{N} | k \in [0...pth.size-1] | \bullet \exists p_2 : PointT | \bullet pointsInPath(pth)[k] = p_2)) \implies true$ —SS]
- exception: $[\neg(validPoints(p_1) \wedge validPoints(p_2)) \implies invalid\_argument)$ —SS]

Local Functions

validRow: $\mathbb{N} \to \mathbb{B}$
[validRow(i) $\equiv 0 < i < nRow = true$, where i $\to \mathbb{N}$ —SS]

validCol: $\mathbb{N} \to \mathbb{B}$
[validCol(i) $\equiv 0 < i < nCol = true$, where i $\to \mathbb{N}$ —SS]

validPoint: PointT $\to \mathbb{B}$
[validPoint(point) $\equiv (point.x \wedge point.y) \geq 0 \wedge (point.x \wedge point.y) < (nRow \wedge nCol) \implies true$, where point $\to$ PointT —SS]

validLine: LineT $\to \mathbb{B}$
[validLine(i) $\equiv (validPoint(l.strt) \wedge validPoint(l.end)) \implies true$, where $l \to$ LineT —SS]

validPath: PathT $\to \mathbb{B}$
[validPath(i) $\equiv \forall : \mathbb{N} | k \in [0...pth.size - 1] \bullet (validLine(pth[k])) \implies true$, where pth $\to$ PathT —SS]

pointsInLine: LineT $\to$ (set of PointT)

pointsInLine (l) [pointsInLine(l) $\equiv i : \mathbb{N} | i \in [0..(l.len-1)] : l.strt.translate($

$l.orient =$ N	$(0, 1)$
$l.orient =$ S	$(0, -1)$
$l.orient =$ W	$(-1, 0)$
$l.orient =$ E	$(1, 0)$

$)$ —SS]
pointsInPath: PathT $\to$ (set of PointT)
[pointsInPath(p) $\equiv \cup(k : \mathbb{N} | k \in [0..(p.size - 1)] : (pointsInLine(p[k])))$, where p $\to$ PathT —SS] pointsInPath(p)

LanduseMap Module

Template Module

LanduseMapT is Seq2D(LanduseT)

DEM Module

Template Module

DEMT is Seq2D($\mathbb{Z}$)

Critique of Design

In general, the specifications were easy to follow. However, one thing that could be added are hints to use the local functions for certain exceptions earlier on in each module because some time was wasted to create exceptions beforehand, only to realize that one could use the local functions as exceptions.