CS 2ME3 Assignment 4, Specification

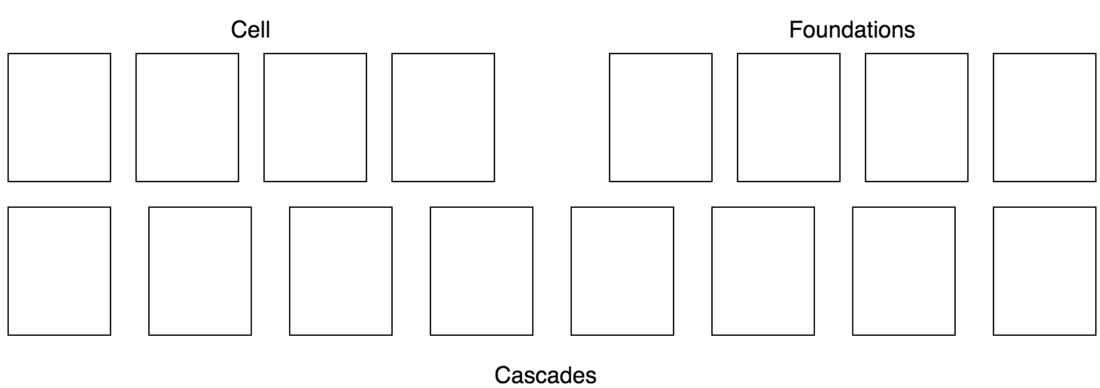
Emily Horsman

April 9, 2018

This document contains a Module Interface Specification for the Model component of the game 'FreeCell'. It assumes that a hypothetical View and Controller component exists but contains no specification for these components. Due to this assumption, there are some access programs in the modules below which are not strictly necessary for the Model MIS, but would be necessary to implement a View and Controller, and happen to be useful for unit testing. These access programs are commented below.

Instead of having a unique data structure (e.g., a Maybe/Optional type) for the free cells, all placements in the game are represented with the same data structure. This is accomplished by having a bounded capacity on this data structure, which is effective because no placement has an infinite bound anyway. Checking this bound is useless on the foundation and cascade placements because the rules for their valid moves would prevent the capacity from ever being exceeded. However, I feel that this bound increases the self-documentation of the instances and is useful for having placements represented homogeneously.

Different sources of the rules use different terminology for each placement in the game. Below is the nomenclature of this document (diagram made with draw.io).



Game Types Module

Module

GameTypes

Syntax

Exported Constants

Ace : RankT = 1
Jack : RankT = 11
Queen : RankT = 12
King : RankT = 13

Exported Types

PlacementT = { Cell, Foundation, Cascade }
SuitT = { Spades, Clubs, Hearts, Diamonds }
RankT = $\{ n : \mathbb{N} \,|\, n \in [1, 13] : n \}$

Semantics

State Variables

None

State Invariant

None

Generic Stack Module

Generic Template Module

StackADT(T)

Uses

N/A

Syntax

Exported Constants

None

Exported Types

Stack(T) = ?

Exported Access Programs

Routine name	In	Out	Exceptions
Stack	$\mathbb{N}$	Stack	invalid_capacity
isEmpty		$\mathbb{B}$	
isFull		$\mathbb{B}$	
capacity		$\mathbb{N}$	
push	T		full
peek		T	empty
pop		T	empty
seq		seq(T)	

[seq() would be required for a hypothetical view. isEmpty() and isFull() violate essentiality given that capacity() and seq() exist, however I believe this violation gives a more understandable design which is an acceptable tradeoff. — EH]

Semantics

State Variables

s: seq of T
capacity: $\mathbb{N}$

State Invariant

None

Assumptions

- The Stack(T) constructor is called for each object instance before any other access routine is called for that object.

Access Routine Semantics

Stack(c):

- transition: $s, \text{capacity} := \langle\rangle\,, c$
- output: $out := self$
- exception: $exc := (c = 0 \Rightarrow \text{invalid\_capacity})$

isEmpty():

- output: $out := |s| = 0$
- exception: None

isFull():

- output: $out := |s| = \text{capacity}$
- exception: None

capacity():

- output: $out := \text{capacity}$
- exception: None

push(v):

- transition: $s := s \,||\, \langle v \rangle$
- exception: $exc := (|s| = \text{capacity} \Rightarrow \text{full})$

peek():

- output: $out := s[|s| - 1]$

- exception: $exc := (|s| = 0 \Rightarrow \text{empty})$

pop():

- transition: $s := s[0..|s| - 2]$
- exception: $exc := (|s| = 0 \Rightarrow \text{empty})$

seq():

- output: $out := s$
- exception: None

Card Module

Template Module

CardADT

Uses

GameTypes for SuitT, RankT

Syntax

Exported Constants

None

Exported Types

CardT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
CardT	SuitT, RankT	CardT	
suit		SuitT	
rank		RankT	
isRed		$\mathbb{B}$	

Semantics

State Variables

s: SuitT
r: RankT

State Invariant

None

Assumptions

- The CardT constructor is called for each object instance before any other access routine is called for that object.

Access Routine Semantics

CardT(S, R):

- transition: $s, r := S, R$
- output: $out := self$
- exception: None

suit():

- output: $out := s$
- exception: None

rank():

- output: $out := r$
- exception: None

isRed():

- output: $out := s \in \{\,\text{Diamonds}, \text{Hearts}\,\}$
- exception: None

Game Module

Template Module

GameADT

Uses

CardADT for CardT, StackADT for Stack, GameTypes for PlacementT, Ace, King

Syntax

Exported Constants

None

Exported Types

GameT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
GameT		GameT	
GameT	seq(Stack(CardT))	GameT	
hasWon		$\mathbb{B}$	
isValidMove	PlacementT, $\mathbb{N}$, PlacementT, $\mathbb{N}$	$\mathbb{B}$	invalid_placement, empty_source
noValidMoves		$\mathbb{B}$	
performMove	PlacementT, $\mathbb{N}$, PlacementT, $\mathbb{N}$		invalid_placement, invalid_move
getCol	PlacementT, $\mathbb{N}$	Stack(CardT)	invalid_placement

Semantics

State Variables

cols: seq of Stack(CardT)

State Invariant

None

Assumptions

- The GameT() constructor is called for each object instance before any other access routine is called for that object.
- Any seq(Stack(CardT)) value passed to the GameT(c) constructor will have been constructed from a previous GameT instance and is thus a valid board.
- Programs using this model specification are aware of the number of cascades, cells, and foundations. invalid_placement will be thrown for an invalid configuration but there is no method to check whether a placement is valid or not because this is considered an axiom of the game and moves can only occur from interactions with the Controller/View.

Access Routine Semantics

GameT():

- transition: $\text{cols} := \text{rng}(\text{possibleCascades}) \,||\, \text{cells} \,||\, \text{foundations}$
  where $\text{cells}, \text{foundations} :=$
  $||(i : \mathbb{N} \,|\, i \in [0..3] : \langle \text{Stack}(1) \rangle), ||(i : \mathbb{N} \,|\, i \in [0..3] : \langle \text{Stack}(13) \rangle)$ [Since the order of the sequence of same-Stack instances does not matter, || can be used as the binary operator of a reduce/fold. — EH]
- output: $out := self$
- exception: None

GameT(c):

- transition: $\text{cols} := c$
- output: $out := self$
- exception: None

hasWon():

- output: $out := \forall (i : \mathbb{N} \,|\, i \in [12..15] :$
  $\neg \text{cols}[i].\text{isEmpty}() \wedge \text{cols}[i].\text{peek}().\text{rank}() = \text{King})$
- exception: None

isValidMove(p, i, q, j):

- output:

		$out :=$
$q = \text{Cell}$		dst.isEmpty()
$q = \text{Foundation}$	$p = \text{Foundation}$	false
	$p \neq \text{Foundation}$	$\text{isValidBuild}(src.\text{peek}(), j)$
$q = \text{Cascade}$		$\text{isValidStack}(src.\text{peek}(), j)$

  where $src, dst := \text{getCol}(p, i), \text{getCol}(q, j)$

- exception: $exc :=$ (
  $\neg\text{isValidPlacement}(p, i) \vee \neg\text{isValidPlacement}(q, j) \Rightarrow \text{invalid\_placement} \,|$
  $\text{getCol}(p, i).\text{isEmpty}() \Rightarrow \text{empty\_source}$
)

noValidMoves():

- output: $out := \neg\, \exists(p, q : \text{PlacementT}, i, j : \mathbb{N} \,|$
  $\text{isValidPlacement}(p, i) \wedge \text{isValidPlacement}(q, j) : \text{isValidMove}(p, i, q, j))$
- exception: None

performMove(p, i, q, j):

- transition: $dst.\text{push}(src.\text{peek}()), src.\text{pop}()$
  where $src, dst := \text{getCol}(p, i), \text{getCol}(q, j)$ [This is an operational specification to keep the spec readable and to avoid violating an interface. — EH]
- exception: $exc :=$ (
  $\neg\text{isValidPlacement}(p, i) \vee \neg\text{isValidPlacement}(q, j) \Rightarrow \text{invalid\_placement} \,|$
  $\neg\text{isValidMove}(p, i, q, j) \Rightarrow \text{invalid\_move}$
)

getCol(p, i):

- output:

	$out :=$
$p = \text{Cascade}$	$\text{cols}[i]$
$p = \text{Cell}$	$\text{cols}[i + 8]$
$p = \text{Foundation}$	$\text{cols}[i + 12]$

- exception: $exc := (\neg\text{isValidPlacement}(p, i) \Rightarrow \text{invalid\_placement})$

Local Functions

isValidPlacement: PlacementT $\rightarrow \mathbb{N} \rightarrow \mathbb{B}$
isValidPlacement$(p, i) \equiv$
$(p = \text{Cell} \lor p = \text{Foundation} \Rightarrow 0 \leq i \leq 3 \,|\, p = \text{Cascade} \Rightarrow 0 \leq i \leq 7)$

isValidBuild: CardT $\rightarrow \mathbb{N} \rightarrow \mathbb{B}$
isValidBuild$(c, j) \equiv ($
$s.\text{isEmpty}() \Rightarrow c.\text{rank}() = \text{Ace} \,|$
$\neg s.\text{isEmpty}() \Rightarrow (s.\text{peek}().\text{suit}() = c.\text{suit}() \land s.\text{peek}().\text{rank}() = c.\text{rank}() - 1)$
)
where $s := \text{getCol}(\text{Foundation}, j)$

isValidStack: CardT $\rightarrow \mathbb{N} \rightarrow \mathbb{B}$
isValidStack$(c, j) \equiv ($
$s.\text{isEmpty}() \,|$
$\neg s.\text{isEmpty}() \Rightarrow (s.\text{peek}().\text{isRed}() \neq c.\text{isRed}() \land s.\text{peek}().\text{rank}() - 1 = c.\text{rank}())$
)
where $s := \text{getCol}(\text{Cascade}, j)$

isDistinct: Stack(CardT) $\rightarrow$ Stack(CardT) $\rightarrow \mathbb{B}$
isDistinct$(a, b) \equiv \neg\exists(i, j : \mathbb{N}, c, c' : \text{CardT} \,|$
$i \in [0..|a.\text{seq}()| - 1] \land j \in [0..|b.\text{seq}()| - 1] \land c = a.\text{seq}()[i] \land c' = b.\text{seq}()[j]$
$: c.\text{suit}() = c'.\text{suit}() \land c.\text{rank}() = c'.\text{rank}()$
)

possibleCascades: seq(Stack(CardT))
possibleCascades $\equiv \{s : \text{seq}(\text{Stack}(\text{CardT})) \,|$
$|s| = 8 \land$
$\forall(i : \mathbb{N} \,|\, i \in [0..7] : s[i].\text{capacity}() = 19) \land$
$\forall(i : \mathbb{N} \,|\, i \in [0..3] : |s[i].\text{seq}()| = 7) \land$
$\forall(i : \mathbb{N} \,|\, i \in [4..7] : |s[i].\text{seq}()| = 6) \land$
$\forall(i, j : \mathbb{N} \,|\, i, j \in [0..7] \land i \neq j : \text{isDistinct}(s[i], s[j]))$
$: s\}$ [This produces a set of all possible board configurations so that one can be chosen at randomly. The range expression of this set comprehension denotes what a valid initial sequence of cascade stacks looks like. — EH]

rng: set(T) $\rightarrow$ T
rng$(s) \equiv$ a random member of the set s with each member having a $1/|s|$ probability of

being chosen