COMP-533 Model-Driven Software Development

Final

Video Slot Machine

Functional Design Models

(40% of final grade)

November 20, 2013

1 Problem Statement

Traditional slot machines (see picture on the left side of Fig. 1) have in recent years been replaced by video slot machines (see picture on the right of Fig. 1). Your task is to elaborate the requirements for a video slot machine, which allows users to gamble credits on spinning reels as described in this document. The requirements described here are taken from a real video slot machine requirements document from industry, but have been shortened to fit the scope of this requirement. In particular, the legislative regulations have been heavily simplified.

2 High-level Game Description

The game is played by wagering credits on the spinning reels. The player can buy credits by inserting coins, bills or tickets in the corresponding acceptors of the Electronic Gaming Machine (EGM) at any time. Credits wagered are subtracted from the player credit meter, and subsequent winnings are added to it. At any time, the current credits meter is displayed on the screen, as well as the credits bet for the current game, the last game outcome, and the credits won in the last game. Before playing, the player can select options such as bet level or which paylines should be active.

When pressing a play button, the game starts and the outcome is decided by a random number generator. This outcome is displayed as spinning reels, that successively stop spinning. Prizes for winning combinations can be looked up on a paytable screen, and some prizes are also displayed in the advertising glass readily visible in the EGM. A special combination awards a host-controlled progressive prize. The player can request to cash his credits out of the machine whenever the reels are not spinning.

If a player ceases to play, i.e., the EGM is idle for a specified amount of time, the machine starts playing an attract animation until a new player inserts a coin or presses one of the buttons.

Figure 1: Mechanical Slot Machine (left) and Video Slot Machine (right)

3 Additional Game Details

3.1 Credits and Promotional Credits

The unit for bets and wins is the "credit". The credit-to-currency conversion ratio can be configured for each EGMs. Typically, a credit is worth \$0.25, but machines allowing higher wagers, such as \$100 per credit, are not uncommon.

Players can buy credits by inserting coins or bills in the corresponding acceptors of the EGM. A special "ticket slot" also accepts tickets issued by a central casino authority that represent a certain amount of money. To validate a ticket, the EGM needs to communicate with the SAS (Slot Accounting System) host. Tickets are gaining popularity and importance, as more and more modern casinos move towards cashless gaming. To attract customers, some casinos also distribute special tickets that contain "promotional credits" that a player can redeem at an EGM to play. These restricted credits, however, can not be converted to currency (cashed out). They must be used for wagering.

Depending on the installed and configured hardware, the current value of credits may be cashed out by a coin hopper, a ticket printer or be hand paid by an attendant. Again, ticket cash out requires a online connection with the SAS host. Winning a prize larger than a certain value forces the prize to be paid by an attendant, who in turn is obliged to notify the IRS (Internal Revenue Service). Winning such a prize suspends the game and puts the EGC in a handpay lockup. Only after the attendant acknowledged the payout and reset the condition (see subsection 3.6.3 below), the game is playable again.

3.2 Reels

There are typically 5 reels in a slot machine, each one with a fixed number of "slots" (typically 22, but there could be more or less). Each slot displays an icon. The visual appearance of the icons is usually aligned with the theme of the EGM. On each reel, a given icon can appear multiple times.

3.3 Paylines

Mechanical slot machines typically only offer one payline: the icons shown in the center of each reel are used to form one 5-icon combination, and then used to lookup the amount won, if any, in the paytable. Video slot machines offer the player the possibility to activate more than one payline. In the stopped position, each reel shows 5 icons (at index -2, -1, 0, 1, and 2). The standard payline looks at the icons at index (0,0,0,0,0). However, additional paylines include the other 4 lines (e.g., -1,-1,-1,-1,-1), or diagonals (e.g. -2, -1, 0, 1, 2), or even waves (0, -1, 0, 1, 0). Fig. 2 shows an example of 30 possible paylines that are available in the Cops n' Bandits video slot machine. Of course, when n paylines are active, each game uses n times the selected wager.

3.4 Out of Service

Any hardware malfunction or possible security threat, along with any maintenance operations, may put the EGM in an "Out of Service" state. While out of service, the EGM is not playable. The attendant light on top of the machine is activated to help the attendant locate the machine that needs service, the SAS is notified, and a window is displayed on the main screen that indicates the out of service condition together with the reason of that condition. The list of reasons that may put EGM in this condition is:

- Maintenance Mode: which can be activated by the assistant on the main assistant screen, or activated remotely by the SAS
- SAS communication error
- Any monitored door is open (main door, card cage door, cash box, belly door or slot door).
- Errors in bill validator: Stacker Full, Bill Jam, Counterfeit bill detected, Bill rejected (reason other than counterfeit bill)
- Errors in printer: Out of paper, Paper low, Carriage jammed
- Errors in coin comparator: Coin in tilt

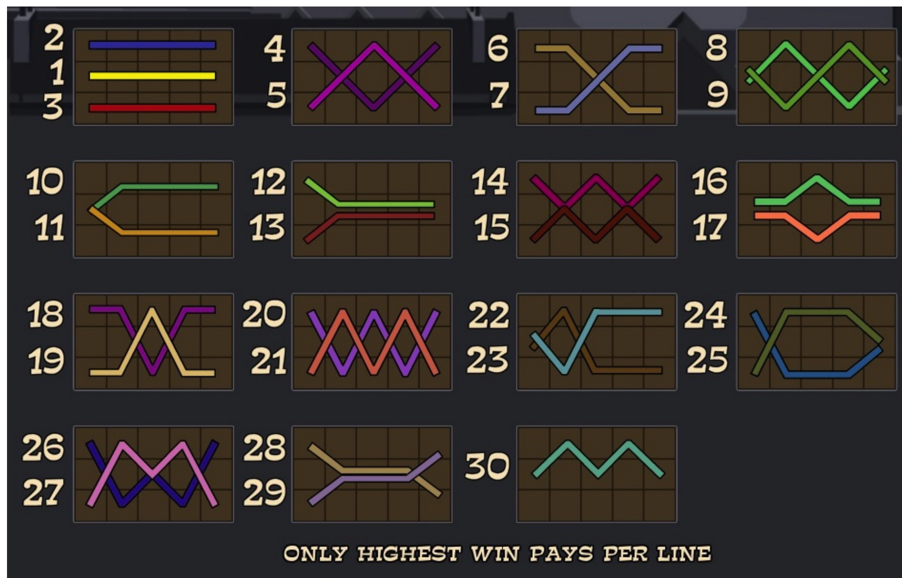

Figure 2: Cops n' Bandits 30 Paylines

Each out of service condition can be reset by fixing the problem (for cases like network error, or door open), or resetting the condition using the option "Return To Service" in the main assistant menu, or resetting it remotely by the SAS. This returns the EGM to normal mode (i.e. the attendant light is switched off and the game is playable again).

3.5 Progressive Jackpot

The EGM can be connected to a progressive prize, i.e., a jackpot shared by a group of EGMs that are connected to the SAS host. If enabled, the EGM transmits upon every bet a fraction of that bet to the SAS host (e.g. 1%), together with the configured group ID for the progressive jackpot, which must match an existing group ID configured in the SAS host and in every other machine contributing to the same jackpot. Periodically, the SAS host communicates the current value of the progressive jackpot to all EGMs, which typically display the current value on screen to stimulate the current player or attract new players.

If a player wins the progressive jackpot, the EGM communicates instantly with the SAS host, which then communicates the final value to the EGM and resets the jackpot.

3.6 Auditing and Configuration

In most countries, stringent laws oversee the construction and use of EGM used for gambling. The following section describes simplified requirements dictated by the Nevada State Gaming Control Board for video slot machines. Access to auditing options is given to casino staff after identifying themselves with attendant or operator keys, both leading the user to the assistant menu screen. If entered using an operator key, a button labeled "Operator" will be displayed, giving the user additional access to a configuration and administrative options screen.

3.6.1 Last Game Events

The main assistant screen gives the auditor access to the last important events that have occurred. Events that are logged are:

- Power on date and time
- Error conditions (see subsection 3.4 above)
- Coins, Bills and Tickets in and out
- Handpays

- Game events with full details (credits bet, paylines selected, reel positions, credits won per payline)

3.6.2 Meters

In addition, the Nevada Technical Standard for Gaming Devices requires the following meters for proper accounting purposes:

- Coin In: The total value of credits bet (corresponds to NV 2.040(a))
- Coin Out: The total value of credits directly paid by the EGM as a result of winning wagers (i.e., excluding hand pays) (corresponds to NV 2.040(b))
- Total Accepted: Total credits inserted into the EGM by any means (coins, bills, tickets). (corresponds to NV 2.040(f) plus NV 2.040(h) plus all accepted tickets)
- Total Jackpot: Total credits paid by an attendant as a result of jackpot handpays (corresponds to NV 2.040(d))
- Total Cancelled: Total value of credits cashed out from the EGM by any means (coins, ticket, handpays not due to a jackpot win) (corresponds to NV 2.040(e) plus NV 2.040(g) plus printed tickets)

You can find details on the meters on the "Proper Accounting for Gaming Devices" document available from the Nevada State Gaming Control Board website: http://gaming.nv.gov/modules/showdocument.aspx?documentid=2918

3.6.3 Handpay Lockups

When the player is awarded a prize greater than or equal to the IRS limit set for the EGM, whether the prize is a progressive or a paytable jackpot, the SAS is notified and game goes into handpay lockup state, not letting the user play until an assistant resets the handpay. When the assistant enters the EGM auditing menu, the main assistant screen is replaced by the handpay screen, where the jackpot is displayed in accounting and player credits. When the assistant performs the handpay, the handpay is registered and the game becomes playable again. The player credits remain as before winning the jackpot.

3.6.4 Operator Menu

An operator, as opposed to an assistant, has access to the operator menu, which gives access to configuration options of the EGM. For instance, the operator can:

- Select the language used for displaying text on screen
- Set the time span after which an idle machine starts playing an attract animation
- Set the IP address of the SAS
- Set the limits for bets, if any
- Set the monetary value of a credit
- Set the PIN and timout value for the demo mode (see subsection 3.8)
- Enable the progressive jackpot and set the percentage of wager transmitted to the SAS
- Print and then reset the meter period used for accounting. The ticket printer is used to output all meter values, and then they are reset and the log is cleared.

3.7 Remote Configuration

The SAS can remotely retrieve auditing information on the EGM (such as meters and last events), as well as set certain configuration parameters. For instance, the SAS can set the EGM time remotely, thus making it possible to synchronize time on all networked EGMs of a casino. Also, the SAS can reset handpays to credits remotely, which relieves the assistant from intervention. In this case, the jackpot amount is added to the player credits.

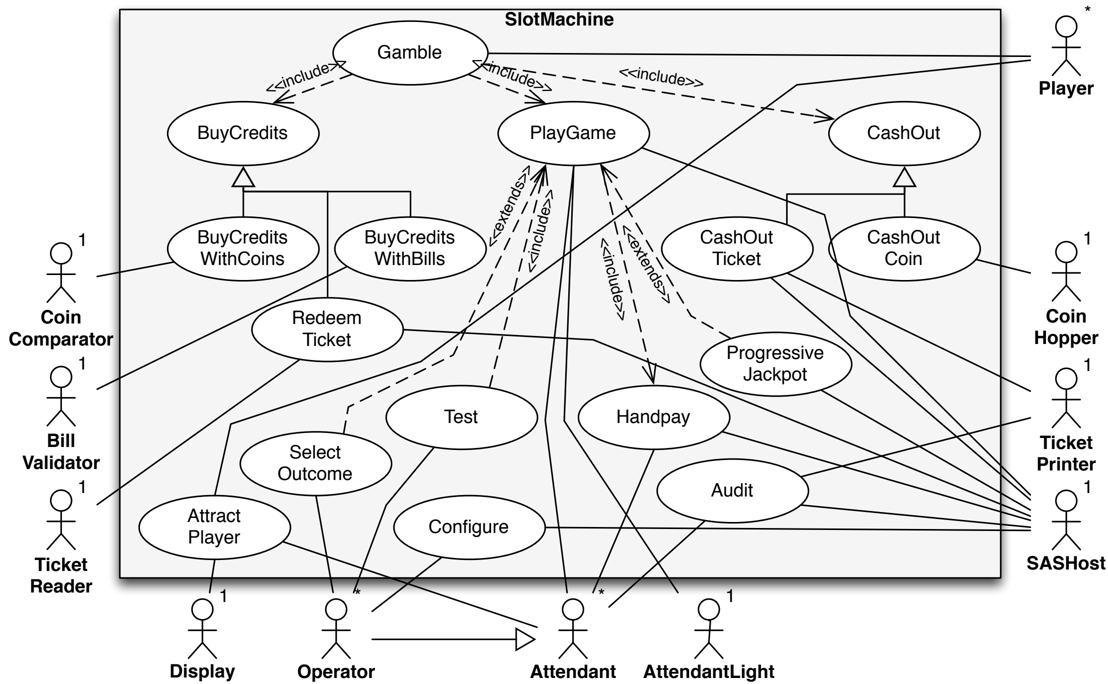

Figure 3: Slot Machine Use Case Diagram

3.8 Demo Mode

For game exhibitions, testing and certification purpose, the EGM has a demo mode, which can be activated by flipping a physical switch that can only be reached after opening the doors of the EGM with an operator key. Additionally, a PIN number needs to be entered. While in demo mode, the operator can add extra credits to the current player meter, or force the winning of some prize. If the operator is inactive for a certain amount of time, the PIN code must be reentered before the demo features can be accessed again. None of the actions performed during the demo mode should affect the meters of the EGM (see subsection 3.6.2). Likewise, when the demo mode is disabled, the current player credits, last game outcome and last credits won should be restored to hold the values that they contained before entering the demo mode.

4 Use Case Model

The use case model of the Slot Machine has already been established. The use cases and their relationships are summarized in figure 3. The individual descriptions of each use case are shown below.

Gamble Use Case

Use Case: Gamble
Scope: EGM
Level: User-goal (could be summary)
Intention in Context: The intention of the Player is to have fun by gambling money on the spinning reels of the EGM, hoping to walk away with more money in the end.
Multiplicity: Only one *Player* can gamble at the EGM at a given time.
Primary Actor: *Player*
Secondary Actor: *AttendantLight*, *SASHost*
Precondition: EGM is in service.
Main Success Scenario:
1. Player buys credits.
   Step 1 can be repeated any number of times, and execute even in parallel with 2.
2. Player plays a game on the EGM.
   Step 2 can be repeated any number of times, until the Player decides to cash out his money.
3. Player cashes out the current credits.

Extensions:
1a. Player is not able to buy credits. Use case ends in failure.
(2-3)a. Player has no credits left. Use case ends in success.
(1-3)a. The *CoinComparator*, *BillValidator*, *TicketReader*, *CoinHopper* or *TicketPrinter* notify the *System* of a problem.
(1-3)a.1. *System* turns on *AttendantLight*.
(1-3)a.2. *System* notifies *SASHost* and goes out of service, waiting for maintenance. Use case ends in failure.

BuyCreditsWithCoins Use Case

Use Case: BuyCreditsWithCoins
Scope: EGM
Level: Subfunction-level (could be user goal)
Intention in Context: The intention of the *Player* is to buy credits for the EGM in exchange of coins.
Multiplicity: Only one *Player* can buy credits at the EGM at a given time.
Primary Actor: *Player*
Secondary Actor: *CoinComparator*
Main Success Scenario:
Step 1 and 2 can be repeated as many times as desired.
1. *CoinComparator* informs *System* that the *Player* inserted a coin.
2. *System* informs *Player* about his current credits.

Extensions:
1a. *CoinComparator* informs *System* of an error. Use case ends in failure.

BuyCreditsWithBills Use Case

Use Case: BuyCreditsWithBills
Scope: EGM
Level: Subfunction-level (could be user goal)
Intention in Context: The intention of the *Player* is to buy credits for the EGM in exchange of bills.
Multiplicity: Only one *Play*er can buy credits at the EGM at a given time.
Primary Actor: *Player*
Secondary Actor: *BillValidator*, *Player*
Main Success Scenario:
Step 1 and 2 can be repeated as many times as desired.
1. *BillValidator* informs *System* that the *Player* inserted a bill.
2. *System* informs *Player* about his current credits.

Extensions:
1a. *BillValidator* informs *System* of an error. Use case ends in failure.
2a. *System does not accept the kind of bill the Player inserted.*
  2a.1 *System* informs *Player* that the inserted bill is not accepted.
  2a.2 *System* commands *BillValidator* to eject the bill. Use case ends in failure.

RedeemTicket Use Case

Use Case: RedeemTicket
Scope: EGM
Level: Subfunction-level (could be user goal)
Intention in Context: The intention of the *Player* is to buy credits for the EGM in exchange of a ticket.
Multiplicity: Only one *Player* can buy credits at the EGM at a given time.
Primary Actor: *Player*
Secondary Actor: *TicketReader*, *SASHost, Player*
Main Success Scenario:

Step 1 and 4 can be repeated as many times as desired.

1. *TicketReader* informs *System* that the *Player* inserted a ticket.
2. *System* contacts *SASHost* to validate ticket.
3. *SASHost* notifies *System* that ticket is valid.
4. *System* informs *Player* about his current credits.

Extensions:
1a. *TicketReader* informs *System* of an error. Use case ends in failure.
3a. *SASHost* notifies *System* that ticked is invalid.
2a.1 *System* informs *Player* that the inserted ticket is not valid.
2a.2 *System* commands *TicketReader* to eject the ticket. Use case continues at step 1.

PlayGame Use Case

Use Case: PlayGame
Scope: EGM
Level: User Goal
Intention in Context: The intention of the Player is to have fun by betting a specific amount of money on the spinning reels, hoping for a good outcome.
Multiplicity: Only one Player can play a game at the EGM at a given time.
Primary Actor: Player
Secondary Actor: *SASHost*, (*PlayButton*, *BetButton*, *TouchScreen* are all facilitators for the *Player*)
Main Success Scenario:

1. *Player* informs *System* about the amount of credits that she wants to bet.
2. *Player* informs the *System* which paylines to use.
   Step 1 and 2 can be done in any order. Step 2 can also be skipped, in which case the previously selected paylines are active.
3. *Player* instructs *System* to start the game.
4. *System* determines the outcome of the game and communicates it to the *Player*.
5. *System* determines the amount of credits won and communicates the amount to the *Player*.

Extensions:
1a || *Player* informs the *System* that he wants to consult the paytable.
1a ||.1 *System* displays paytable for *Player*. Use case continues at step 1.
5a. *System is not in demo mode and determines that the Player has won the progressive jackpot.*
5a.1 *System* informs *SASHost* that the successive jackpot was won.
5a.2 *SASHost* informs *System* of the amount that was won. Use case continues at step 5.
5b. *System is not in demo mode and determines that the amount won is above or equal to the IRS win limit.*
5b.1 *System* performs handpay.

Handpay Use Case

Use Case: Handpay
Scope: EGM
Level: Subfunction-level
Intention in Context: The player has won a sum of money that is above the IRS win limit. The sum needs to be paid by handpay.
Multiplicity: Only one handpay can be active at a given time.
Primary Actor: IRS (is actually a stakeholder that does not directly interact with system)
Secondary Actor: *SASHost*, *AttendantLight (Touchscreen* is a facilitator actor for *Attendant* / *Operator*)
Main Success Scenario:
1. *System* turns on *Attendant Light*.
2. *System* notifies *SASHost*.
3. *Attendant* or *Operator* identifies himself to System.
4. *Attendant* notifies *System* that handpay has been made.
5. *System* turns off *AttendantLight*.

Extensions:
3a *SASHost* instructs *System* to reset the win to credits. Use case continues at step 5.

CashOutCoins Use Case

Use Case: CashOutCoins
Scope: EGM
Level: Subfunction-level (could be user goal)
Intention in Context: The intention of the *Player* is to exchange his credits for coins.
Multiplicity: Only one *Player* can cashout credits at the EGM at a given time.
Primary Actor: *Player*
Secondary Actor: *CoinHopper*, *Touchscreen* (facilitator actor for *Player*)
Precondition: The coin hopper contains enough coins to allow the player to cash out his credits.
Main Success Scenario:
Step 1 and 2 can be repeated as many times as desired.
1. *Player* informs *System* that he wants to cash out his credits as coins.
2. *System* requests *CoinHopper* to eject the amount of cashable credits in coins.

Extensions:
2a. *CoinHopper* informs *System* about coin jam. Use case ends in failure.

CashOutTicket Use Case

Use Case: CashOutTicket
Scope: EGM
Level: Subfunction-level (could be user goal)
Intention in Context: The intention of the *Player* is to exchange his credits for a ticket.
Multiplicity: Only one *Player* can cashout credits at the EGM at a given time.
Primary Actor: *Player*
Secondary Actor: *TicketPrinter*, *Player*, *SASHost*
Main Success Scenario:
1. *Player* informs *System* that he wants to cash out his credits as a ticket.
2. *System* requests ticket with the correct amount from *SASHost*.
3. *SASHost* sends *System* a ticket.
4. *System* instructs *TicketPrinter* to print the ticket.

Extensions:
3a. *System can not communicate with the SASHost.* Use case ends in failure.
5a. *TicketPrinter* informs *System* about an error. Use case ends in failure.

Audit Use Case

Use Case: Audit
Scope: EGM
Level: User Goal
Intention in Context: The intention of the *Auditor* (*Attendant*, *Operator* or *SASHost*) is to query information about the history of events registered in the EGM.
Multiplicity: Only one Attendant or *Operator* can audit the EGM at a given time, and only if no player is currently gambling on the machine. The *SASHost* can audit the EGM while a player is gambling on the machine.
Primary Actor: *Attendant*, *Operator* or *SASHost* (referred to as *Auditor*)
Secondary Actor: *TicketPrinter*
Main Success Scenario:
1. *Auditor* identifies himself to *System*.
2. *System* acknowledges successful identification to *Auditor*.
   Steps 3 and 4 can be repeated as often as requested by the Auditor.
3. *Auditor* requests information about a meter or event history from *System*.
4. *System* presents requested information to *Auditor*.
5. *Auditor* logs out of *System*.

Extensions:
1a. *Auditor* is an SASHost. Identification is not necessary, as it is peformed by the underlying communication protocol. Use case continues at step 3.
5a. *Auditor* (which must be an *Attendant* or *Operator*) requests *System* to printout the history and meter values and reset the machine.
5a.1 *System* prints event history and meters on *TicketPrinter*. Use case continues at step 5.
5a.1a. *TicketPrinter* informs *System* about problem. Use case continues at step 5 without resetting the machine.

Configure Use Case

Use Case: Configure
Scope: EGM
Level: User Goal
Intention in Context: The intention of the *Operator* or *SASHost* is to set certain configuration parameters (which include current time, current language, creditToCurrencyRatio, idleTimeBeforeAnimation, demoPIN, demoTimeout, maxBetCredits) in the EGM.
Multiplicity: Only one *Operator* can set the EGM configuration parameters at a given time, and only if no player is currently gambling on the machine. The *SASHost* can set certain configuration parameters (e.g. the current time) while a player is gambling on the machine.
Primary Actor: *Operator* or *SASHost*
Secondary Actor: n/a
Main Success Scenario:
1. *Operator* identifies himself to *System*.
2. *System* acknowledges successful identification to *Operator*.
   Steps 3 and 4 can be repeated as often as requested by the Auditor.
3. *Operator* notifies *System* about a configuration parameter that is to be set.
4. *System* acknowledges setting of configuration parameter to *Operator*.
5. *Operator* logs out of *System*.

Extensions:
1a. Configuration is performed by an *SASHost*. Identification is not necessary, as it is peformed by the underlying communication protocol. Use case continues at step 3.
5a. Configuration is performed by an *SASHost*. Logout is unnecessary. Use case ends in success.

Test Use Case

Use Case: Test
Scope: EGM
Level: User Goal
Intention in Context: The intention of the *Operator* is to test the EGM by running it in demo mode.
Multiplicity: Only one *Operator* can test the EGM at a given time, and only if no player is currently gambling on the machine.
Primary Actor: *Operator*
Secondary Actor: n/a
Main Success Scenario:
1. *Operator* identifies himself to *System*.
2. *System* acknowledges successful identification to *Operator*.
3. *Operator* instructs *System* to switch to demo mode.
4. *System* prompts *Operator* for demo PIN.
5. *Operator* inputs demo PIN into *System*.
6. *System* acknowledges demo mode switch to *Operator*.
   Step 7 and 8 can be repeated as many times as needed.
7. *Operator* informs *System* about the amount of demo credits to add.
8. *System* acknowledges the adding of credits to *Operator*.
   Step 9 can be repeated as many times as needed.
9. *Operator* plays the game.
10. *Operator* informs *System* that he wants to exit the demo mode.
11. *Operator* logs out of *System*.

Extensions:
(6-10)a. *System was idle for longer than demoTimeout. Use case continues at step 4.*

ServiceMachine Use Case

Use Case: ServiceMachine
Scope: EGM
Level: User Goal
Intention in Context: The intention of the *Attendant* is to resolve the error condition that prevents the machine from normal operation.
Multiplicity: Only one *Attendant* can be servicing the machine at a given time.
Primary Actor: *Attendant*
Secondary Actor: *AttendantLight*, *SASHost*
Main Success Scenario:
1. *Attendant* identifies himself to *System*.
2. *System* acknowledges successful identification to *Operator*.
3. *Attendant* notifies *System* that a specific error condition has been resolved.
4. *System* acknowledges error resolution to *Operator*.
5. *Attendant* logs out of *System*.
6. *System* turns off *AttendantLight*.

Extensions:
1a. *SASHost* commands *System* to ignore the error. Use case continues at step 4.

AttractPlayer Use Case

Use Case: AttractPlayer
Scope: EGM
Level: User Goal
Intention in Context: The intention of the *Casino* is to attract a new player to the machine.
Multiplicity: Only one AttractPlayer animation can be running at a given time.
Primary Actor: *Casino* (is a stakeholder that does not interact with the system directly)
Secondary Actor: *Player*, *Attendant*
Main Success Scenario:

System is idle for a specific amount of time (timeBeforeIdleAnimation).
1. *System* plays attract animation on *TouchScreen*.
2. *Player* notifies *System* of his presence.

Extensions:
1a. *SASHost* notifies *System* of current value of progressive jackpot.
1a.1. *System* advertizes value of progressive jackpot to *Player*. Use case continues at step 1.
2a. *Attendant* notifies *System* of his presence. Use case ends in failure.

5 Environment Model

Type Definitions

The environment model (see figure 4) and concept model (see figure 5) of the slot machine assume the existence of the following types:

- `Credit`: a type encoding a non-negative integral number.
- `Money`: a type encoding an amount of money in cents.
- `type Percentage is Real range 0..1`
- `type SlotRange is Integer range 0..22`
- `type OutcomeType is TupleType{reel1pos: SlotRange, reel2pos: SlotRange, reel3pos: SlotRange, reel4pos SlotRange, reel5pos: SlotRange}`
- `type OffsetType is TypleType{offset1: Integer, offset2: Integer, offset3: Integer, offset4: Integer, offset5: Integer}`
- `type SlotMachineMode is enum {play, maintenance, handpay, demo, outOfService, idle}`
- `type CashoutKind is enum {coin, ticket}`
- `type PlayerErrorNotificationKind is enum {billValidationError, ticketValidationError, ...}`
- `type AuditKind is enum {coinInMeter, coinOutMeter, totalJackpotMeter, lastGames, ...}`
- `type ServicePersonKind is enum {attendant, operator, unknown}`
- `type Language is enum {english, french, german, bengali, kanji, mandarin, ...}`
- `type CoinComparatorStatus is enum {ok, coinTilt}`
- `type BillValidatorStatus is enum {ok, stackerFull, billJam, counterfeitBill, billRejected}`
- `type TicketReaderStatus is enum {ok, ticketJam}`
- `type CoinHopperStatus is enum {ok, coinJam}`
- `type TicketPrinterStatus is enum {ok, paperLow, outOfPaper, carriageJam}`
- `type SASCommunicationStatus is enum {ok, communicationError}`
- `type ErrorKind is enum` {union of all actor status enums above}
- `type MeterKind is enum {CoinIn, CoinOut, TotalAccepted, TotalJackpot, TotalCancelled}`

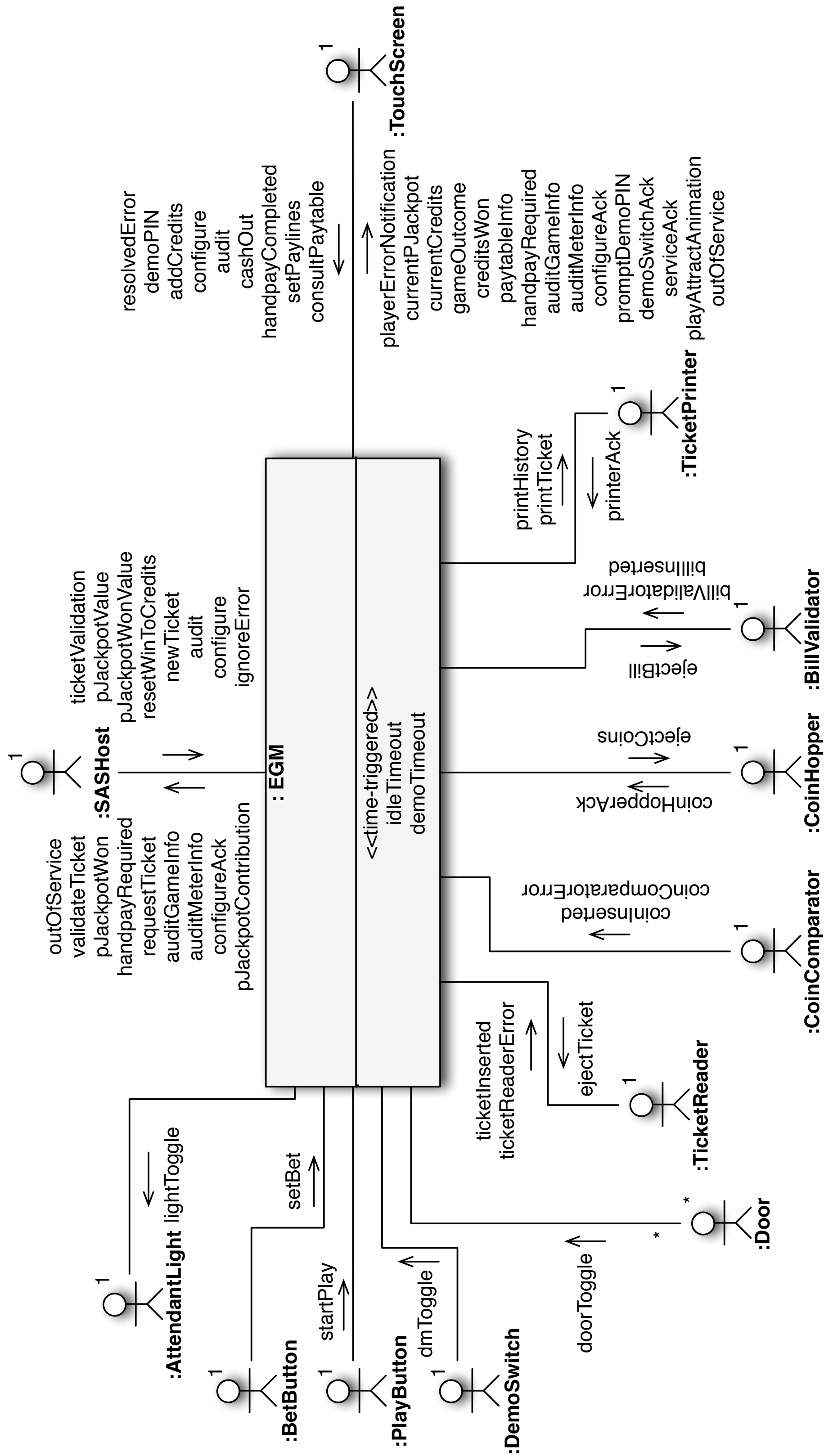

Figure 4: Slot Machine Environment Model

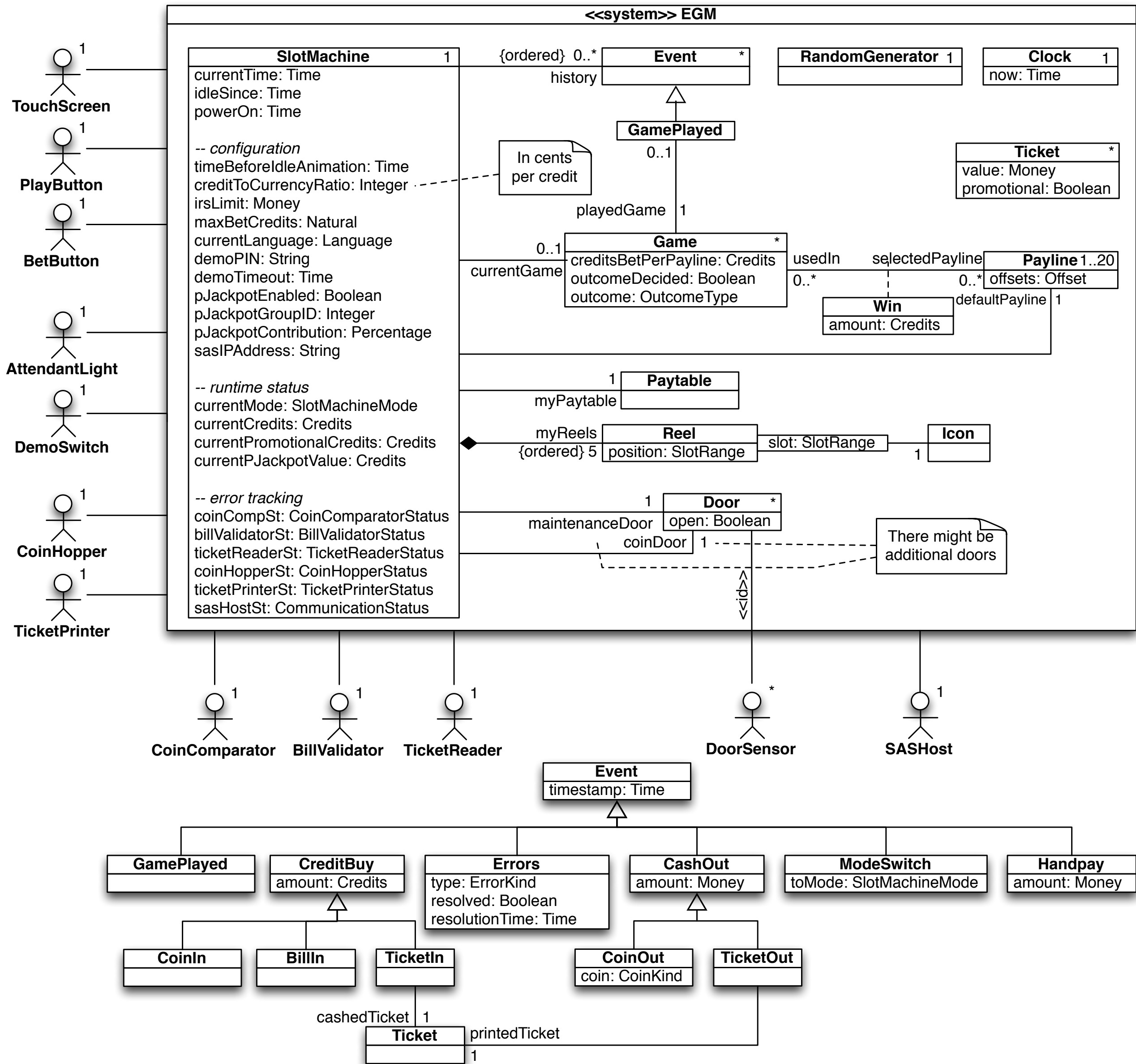

Figure 5: Slot Machine Concept Model

Input Messages

- addCredits(cr: Credits: sent by *TouchScreen* when operator adds credits in demo mode.
- audit(a: AuditKind): sent by *TouchScreen* when *Attendant* or *Operator* wants to query the meters or the recent events of the slot machine. Can also be sent by *SASHost*.
- billInserted(m: Money): sent by *BillValidator* when *Player* inserts a bill.
- billValidatorError(s: BillValidatorStatus): sent by *BillValidator* when an error occurs.
- cashOut(c: CashoutKind): sent by *TouchScreen* when *Player* wants to cash out his credits.
- coinComparatorError(s: CoinComparatorStatus): sent by *CoinComparator* when an error occurs.
- coinHopperAck(s: CoinHopperStatus): sent by *CoinHopper* when printing fails for some reason.
- coinInserted(m: Money): sent by *CoinSlot* when *Player* inserts a coin.
- configure(..): sent by *TouchScreen* when *Operator* wants to change the configuration of the slot machine. The parameters must clearly identify the configuration parameter that is requested to be modified and the value to which it should be set. Can also be sent by *SASHost*.
- consultPaytable(): sent by *Touchscreen* when *Player* wants to look at the winning combinations.
- demoPIN(pin: String): sent by *TouchScreen* when operator provides the demo PIN.
- dmToggle(on: Boolean): sent by *DemoSwitch* when operator activates or deactivates the demo mode.
- doorToggle(who: ServicePersonKind, open: Boolean): sent by *Door* when opened or closed.
- handpayCompleted(): sent by *Touchscreen* when *Attendant* or *Operator* has completed the handpay.
- ignoreError(e: ErrorStatus): sent by *SASHost* to instruct the system to ignore an error.
- newTicket(t: Ticket): sent by *SASHost.*
- pJackpotValue(m: Money): sent by *SASHost* to inform EGM of the current value of the progressive jackpot.
- pJackpotWonValue(t: Time, m: Money): sent by *SASHost* to inform EGM of the value of the progressive jackpot when it was won at time t.
- printerAck(e: PrinterStatus): sent by *TicketPrinter* when printing fails for some reason.
- resetWinToCredits(): sent by *SASHost* to reset the handpay win to credits.
- resolvedError(e: ErrorStatus): sent by *TouchScreen* when attendant or operator has serviced the machine.
- setBet(c: Credits): sent by *BetButton* when *Player* determines the amount of credits to bet.
- setPaylines(p: Set(Payline)): sent by *Touchscreen* when *Player* selects desired paylines.
- startPlay(): sent by *PlayButton* when *Player* wants to start spinning the reels.
- ticketInserted(t: Ticket): sent by *TicketReader* when *Player* inserts a ticket.
- ticketReaderError(s: TicketReaderStatus): sent by *TicketReader* when an error occurs.
- ticketValidation(t: Ticket, valid: Boolean): sent by *SASHost* to inform EGM about validity of ticket t.
- <<time-triggered>> demoTimeout(): triggered when the system has been in demo mode and idle for a duration of *demoTimeout*.
- <<time-triggered>> idleTimeout(): triggered when the system has been idle for a duration of *timeBeforeIdleAnimation*.

Output Messages

- `auditGameInfo(games: Sequence(Game))`: sent to *TouchScreen* or to *SASHost* to communicate the game information requested with a previous audit message
- `auditMeterInfo(m: MeterKind, value: Integer)`: sent to *TouchScreen* or to *SASHost* to communicate the information requested with a previous audit message
- `configureAck(..)`: sent to *TouchScreen* or *SASHost* to acknowledge setting of configuration parameter.
- `creditsWon(winAmount: Credits, currentCredits: Credits)`: sent to *TouchScreen* to inform *Player* of winnings.
- `currentCredits(c: Credits)`: sent to *TouchScreen* to inform *Player* of the number of credits currently available for betting.
- `currentPJackpot(m: Money)`: sent to *TouchScreen* to inform *Player* of the current value of the progressive jackpot.
- `demoSwitchAck()`: sent to *TouchScreen* to acknowledge entering of demo mode.
- `ejectBill()`: sent to *BillValidator* when EGM wants to eject the current bill.
- `ejectCoins(m: Money)`: sent to *CoinHopper* when a *Player* wants to cash out with coins.
- `ejectTicket()`: sent to *TicketReader* when EGM wants to eject the current ticket.
- `gameOutcome(o: Outcome)`: sent to *TouchScreen* to inform *Player* of game outcome.
- `handpayRequired(amount: Money)`: sent to *SASHost* and *TouchScreen* when a *Player* wins a substantial amount of money.
- `outOfService()`: sent to *SASHost* and *TouchScreen* when EGM goes out of service.
- `paytableInfo(p: Paytable)`: sent to *TouchScreen* when player wants to consult the paytable.
- `pJackpotContribution(t: Time, m: Money)`: sent to *SASHost* when a *Player* spins the reels.
- `pJackpotWon(t: Time)`: sent to *SASHost* when a *Player* wins the progressive jackpot.
- `playAttractAnimation(..)`: sent to *TouchScreen* when system has been idle for a long time.
- `playerErrorNotification(e: PlayerErrorNotificationKind)`: sent to *TouchScreen* to inform *Player* of a problem.
- `printHistory(h: Sequence(Event))`: sent to *TicketPrinter* when *Operator* requests to reset the machine before erasing the event history.
- `printTicket(t: Ticket)`: sent to *TicketPrinter* during cash out.
- `promptDemoPIN()`: sent to *TouchScreen* to request demo PIN from *Operator*.
- `requestTicket(amount: Money)`: sent to *SASHost* when a *Player* wants to cash out with a ticket.
- `serviceAck()`: sent to *TouchScreen* to acknowledge service operation.
- `toggleLight(on: Boolean)`: sent to *AttendantLight* to turn it on or off.
- `validateTicket(t: Ticket)`: sent to *SASHost* when a *Player* wants to redeem a ticket.
- `welcome(operator: Boolean)`: sent to *TouchScreen* to greet the *Operator* or *Attendant*.

6 Operation Model

A partial Operation Model for the EGM is given below. It includes among the system operations specifying the behaviour that achieves the Gamble use case also a partial definition of the functionality provided by the audit system operation. The Operation Model assumes the existance of the following OCL functions:

- `RandomGenerator::random() : SlotRange` – a function that yields a random slot position.
- `Outcome::offsetBy(o: Offset) : Outcome` – a function that offsets a game outcome by an offset o to yield a new game outcome
- `Paytable::lookupWin(o: Outcome) : Credits` – a function that maps a game outcome to the number of credits that are awarded for that outcome. In the case where the outcome wins the progressive jackpot, the function returns 0 (the number of credits won has to be queried from the SASHost).
- `Paytable::winsJackpot(o: Outcome) : Boolean` – a function that determines if the game outcome o corresponds to winning the jackpot of the machine
- `Paytable::winsPJackpot(o: Outcome) : Boolean` – a function that determines if the game outcome o corresponds to winning the progressive jackpot

Operation: EGM::coinInserted(m: Money);
Description: The EGM receives a notification that the player inserted a coin into the coin comparator.
Scope: SlotMachine, CoinIn, Clock;
New:
newCoinInEvent: CoinIn;
Messages:
TouchScreen::{CurrentCredits}
Post:
self.currentCredits = **self**.currentCredits**@pre** + m / **self**.slotMachine.creditToCurrencyRatio &
newCoinInEvent.**oclIsNew**() &
newCoinInEvent.timestamp = **self**.clock.now &
newCoinInEvent.amount = m / **self**.slotMachine.creditToCurrencyRatio &
self.slotMachine.history = **self**.slotMachine.history**@pre**→**append**(newCoinInEvent) &
self.touchScreen^currentCredits(**self**.currentCredits + **self**.currentPromotionalCredits)

Operation: EGM::coinComparatorError(s : CoinComparatorStatus)
Description: The EGM receives a notification that an error occurred when reading the coin inserted by the player.
Scope: SlotMachine, Clock, Errors;
New:
errorEvent : Errors;
modeSwitchEvent : ModeSwitch;
Messages: TouchScreen::{playerErrorNotification, OutOfService}; SASHost::{OutOfService}; AttendantLight::{Toggle}
Post:
self.slotMachine.coinCompSt = s &
self.errorLogged(s) &
self.isOutOfService()

Predicate: EGM::isOutOfService();
New:
modeSwitchEvent : ModeSwitch;
Messages: TouchScreen::{OutOfService}; SASHost::{OutOfService}; AttendantLight::{Toggle}
Body:
self.attendantLight^toggle(true) &
self.sasHost^outOfService() &
self.touchScreen^outOfService() &

self.slotMachine.currentMode = SlotMachineMode::OutOfService &
modeSwitchEvent.**oclIsNew**(timestamp = **self**.clock.now, toMode = SlotMachineMode::outOfService,) &
self.slotMachine.history→**includes**(modeSwitchEvent);

Predicate: EGM::errorLogged(e: ErrorKind);
New:
errorEvent : Errors;
Messages: TouchScreen::{playerErrorNotification};
Body:
self.touchScreen^playerErrorNotification(e) &
errorEvent.**oclIsNew**(timestamp = **self**.clock.now, type = e, resolved = **false**, resolutionTime = ?) &
self.slotMachine.history→**includes**(errorEvent);

Operation: EGM::ticketInserted(t: Ticket);
Scope: SlotMachine, Ticket;
Messages:
SASHost::{ValidateTicket};
Post:
self.SASHost^validateTicket(t)

Operation: EGM::ticketValidation(t: Ticket, valid: Boolean, m: Money);
Scope: SlotMachine; TicketIn; Clock
New:
newTicketInEvent: TicketIn;
errorEvent: Errors;
Messages:
TouchScreen::{CurrentCredits, PlayerErrorNotification}
TicketReader::{EjectTicket}
Post:
if valid **then**
if t.promotional **then**
self.currentPromotionalCredits = **self**.currentPromotionalCredits**@pre** + m / **self**.slotMachine.creditToCurrencyRatio
else
self.currentCredits = **self**.currentCredits**@pre** + m / **self**.slotMachine.creditToCurrencyRatio
endif &
newTicketInEvent**.oclIsNew**() &
newTicketInEvent.timestamp = **self**.clock.now &
newTicketInEvent.amount = m / **self**.slotMachine.creditToCurrencyRatio &
newTicketInEvent.cashedTicket = t &
self.slotMachine.history = **self**.slotMachine.history**@pre**→**append**(newTicketInEvent) &
self.touchScreen^currentCredits(**self**.currentCredits + **self**.currentPromotionalCredits)
else
self.touchScreen^playerErrorNotification(PlayerErrorNotificationKind::ticketValidationError) &
self.ticketReader^ejectTicket()
endif

Operation: EGM::ticketReaderError(e: TicketReaderStatus);
Scope: SlotMachine, Ticket;
Messages:
TicketReader::{EjectTicket}, TouchScreen::{PlayerErrorNotification, OutOfService}, SASHost::{OutOfService}, AttendantLight:{Toggle};
Post:
if e = TicketReaderStatus::TicketJam **then**
self.ticketReader^ejectTicket() &

self.touchScreen^playerErrorNotification(PlayerErrorNotificationKind::ticketReaderError)
endif &
self.slotMachine.ticketPrinterSt = e &
self.errorLogged(e) &
self.isOutOfService()

Operation: EGM::setBet(c: Credits);
Scope: SlotMachine, Game; GamePlayed, Cashout
Alias:
previousGame: Game = **self**.slotMachine.history→**select**(e: Event | e.**oclIsKindOf**(GamePlayed))→**last()**.playedGame**;**
lastCashout: CashOut = **self**.slotMachine.history→**select**(e: Event | e.**oclIsKindOf**(CashOut))→**last();**
New:
newGame: Game;
Pre:
c <= **self**.maxBetCredits
Post:
if (**self**.slotMachine.currentGame**@pre**→**isEmpty**() **or self**.slotMachine.currentGame.outcomeDecided) **then**
newGame.**oclIsNew**() &
newGame.**all** = **Tuple**{c, **false**, ?} &
self.slotMachine.currentGame = newGame &
if not previousGame.**oclIsUndefined**() **and** (lastCashout.**oclIsUndefined**() **or** lastCashout.timestamp < previousGame.timestamp) **then**
newGame.selectedPaylines = previousGame.selectedPaylines
else
newGame.selectedPaylines = **self**.slotMachine.defaultPayline
endif
else
self.slotMachine.currentGame.creditsBetPerPayline = c
endif

Operation: EGM::setPaylines(pl: Set(Payline));
Scope: SlotMachine, Game, Payline;
New:
newGame: Game;
Post:
if (**self**.slotMachine.currentGame**@pre**→**isEmpty**() **or self**.slotMachine.currentGame.outcomeDecided) **then**
newGame.**oclIsNew**() &
newGame.**all** = **Tuple**{?, **false**, ?} &
self.slotMachine.currentGame = newGame &
end &
self.slotMachine.currentGame.selectedPaylines = pl

Operation: EGM::startPlay();
Scope: Game; SlotMachine, Reel, RandomGenerator, GamePlayed, Payline
Messages: TouchScreen::{CreditsWon, GameOutcome, HandpayRequired}, SASHost::{PJackpotContribution, PJackpotWon, HandpayRequired}, AttendantLight::{Toggle}
Alias:
currentGame: Game = **self**.slotMachine.currentGame**@pre**;
demoEnabled: Boolean = **(self**.slotMachine.currentMode**@pre** = SlotMachineMode::demoMode);
pTable = **self**.slotMachine.myPaytable;
totalBet: Credits = currentGame.creditsBetPerPayline * currentGame.selectedPaylines→**size**();
totalPromotionalCreditsUsed: Credits =
if self.currentPromotionalCredits**@pre** >= totalBet **then**
totalBet

```
    else
      self.currentPromotionalCredits@pre
    endif;
   totalCreditsUsed: Credits = totalBet - totalPromotionalCreditsUsed;
   totalWin: Credits = currentGame.win.amount→sum();
   pJackpotWasWon = currentGame.win→exists(w: Win | self.slotMachine.myPaytable.winsPJackpot
       (currentGame.outcome.offsetBy(w.selectedPayline.offsets))) and not demoEnabled;
  New:
   playEvent: GamePlayed;
  Post:
   – game outcome has been determined, stored and communicated
   currentGame.outcomeDecided &
   currentGame.outcome = self.randomGenerator.random() &
   self.slotMachine.myReels→at(1).position = currentGame.outcome.reel1pos &
   self.slotMachine.myReels→at(2).position = currentGame.outcome.reel2pos &
   self.slotMachine.myReels→at(3).position = currentGame.outcome.reel3pos &
   self.slotMachine.myReels→at(4).position = currentGame.outcome.reel4pos &
   self.slotMachine.myReels→at(5).position = currentGame.outcome.reel5pos &
   self.touchScreen^gameOutcome(currentGame.outcome) &
   – game is recorded if we're not in demo mode
   if not demoEnabled then
     playEvent.oclIsNew() &
     playEvent.playedGame = currentGame
   endif &
   – promotional credits have been used, if possible
   self.currentPromotionalCredits = self.currentPromotionalCredits@pre - totalPromotionalCreditsUsed &
   – winnings have been looked up
   currentGame.win→forAll(w: Win | w.amount = pTable.lookupWin(currentGame.outcome.offsetBy
     (w.selectedPayline.offsets))) &
   – handling of progressive jackpot
   if self.slotMachine.pJackpotEnabled and not demoEnabled then
     self.sasHost^pJackpotContribution(self.slotMachie.currentTime, self.slotMachine.creditToCurrencyRatio *
       totalBet * self.slotMachine.pJackpotContribution
   endif &
   if pJackpotWasWon and not demoEnabled then
     self.sasHost^pJackpotWon(self.slotMachine.currentTime) &
     self.slotMachine.currentCredits = self.slotMachine.currentCredits@pre - totalCreditsUsed
   else
     if totalWin >= self.slotMachine.irsLimit / self.slotMachine.creditToCurrencyRatio and not demoEnabled then
       – initiated handpay, if needed
       self.slotMachine.currentCredits = self.slotMachine.currentCredits@pre - totalCreditsUsed
       self.switchedToHandpay(total * self.slotMachine.creditToCurrencyRatio)
     else
       self.slotMachine.currentCredits = self.slotMachine.currentCredits@pre - totalCreditsUsed + totalWin &
      self.touchScreen^creditsWon(totalWin, self.slotMachine.currentCredits + self.slotMachine.currentPromotionalCredits)
&
     endif
   endif

Predicate: EGM::switchedToHandpay(amount: Money);
  New:
   modeSwitchEvent: ModeSwitch;
  Messages: TouchScreen::{HandpayRequired}; SASHost::{HandpayRequired}; AttendantLight::{Toggle}
  Body:
```

self.attendantLight^toggle(**true**) &
self.sasHost^handpayRequired(amount) &
self.touchScreen^handpayRequired(amount) &
self.slotMachine.currentMode = SlotMachineMode::handpay &
modeSwitchEvent.**oclIsNew**(timestamp = **self**.clock.now, toMode = SlotMachineMode::handpay) &
self.slotMachine.history→**includes**(modeSwitchEvent);

Operation: EGM::pJackpotWonValue(t: Time, m: Money);
Scope: Game, SlotMachine;
Messages: TouchScreen::{CreditsWon, HandpayRequired}, AttendantLight::{Toggle}, SASHost::{HandpayRequired}
Alias:
currentGame: Game = **self**.slotMachine.currentGame**@pre**;
totalWin: Credits = currentGame.win.amount→**sum**();
pJackpotWin: Win = currentGame.win→**any**(w | **self**.slotMachine.myPaytable.winsPJackpot
(w.usedIn.outcome.offsetBy(w.selectedPayline.offsets))));
Post:
pJackpotWin.amount = m / **self**.slotMachine.creditToCurrencyRatio &
if totalWin >= **self**.slotMachine.irsLimit / **self**.slotMachine.creditToCurrencyRatio **and not** demoEnabled **then**
self.switchedToHandpay(totalWin * **self**.slotMachine.creditToCurrencyRatio)
else
self.slotMachine.currentCredits = **self**.slotMachine.currentCredits**@pre** + totalWin &
self.touchScreen^creditsWon(totalWin,
self.slotMachine.currentCredits + **self**.slotMachine.currentPromotionalCredits) &
endif

Operation: EGM::handpayCompleted();
Scope: SlotMachine;
New:
modeSwitchEvent: ModeSwitch;
Messages: AttendantLight::{Toggle}
Pre:
self.slotMachine.currentMode = SlotMachineMode::handpay
Post:
self.switchedToPlay();

Predicate: EGM::switchedToPlay();
New:
modeSwitchEvent: ModeSwitch;
Messages: AttendantLight::{Toggle}
Body:
self.attendantLight^toggle(**false**) &
self.slotMachine.currentMode = SlotMachineMode::play &
modeSwitchEvent.**oclIsNew**(timestamp = **self**.clock.now, toMode = SlotMachineMode::play) &
self.slotMachine.history→**includes**(modeSwitchEvent);

Operation: EGM::resetToCredits();
Scope: SlotMachine, Paytable;
Messages: AttendantLight::{Toggle}
Alias:
totalWin: Credits = currentGame.win.amount→**sum**();
Post:
self.switchedToPlay() &
self.slotMachine.currentCredits = **self**.slotMachine.currentCredits**@pre** + totalWin

Operation: EGM::consultPaytable();
  Scope: SlotMachine, Paytable;
  Messages: TouchScreen::{PaytableInfo}
  Post:
   self.touchScreen^paytableInfo(**self**.slotMachine.myPaytable)

Operation: EGM::audit(a: AuditKind);
  Scope: SlotMachine, GameEvent, Game, Paytable, Win;
  Messages: SASHost::{AuditGameInfo, AuditMeterInfo}
  Alias:
   lastCashout: **self**.slotMachine.history→**select**(oclIsKindOf(CashOut))→**sortedBy(**timestamp)→**last**()
   eventsForThisPlayer: Set(Event) =
    if lastCashout.**oclIsUndefined**() **then**
     self.slotMachine.history
    else
     self.slotMachine.history→**select(**e | e.timestamp > lastCashout.timestamp)
    endif
  Post:
   if a = AuditKind::lastGames **then**
    sender^auditGameInfo(eventsForThisPlayer→**select**(**oclIsKindOf**(GamePlayed))
   elsif a = AuditKind::totalJackpotMeter then
    sender^AuditMeterInfo**(self**.slotMachine.history→**select**(e | e.**oclIsKindOf**(GamePlayed)).playedGame.win→**select**
     (w | **self**.slotMachine.myPaytable.winsJackpot(w.usedIn.outcome.offsetBy(w.selectedPayline.offsets))).amount→**sum();**
   endif

7 Design

In this part you are to elaborate a partial design for the EGM System. Some design decisions have already been made by some higher power :)

- The paylines that a player can select (i.e., how many paylines the EGM offers and what offsets each payline has) are hard-coded, i.e. it is not possible to change the offsets of the paylines when configuring the EGM. As a result, it is possible to identify a payline simply by a number between 1 and 20.
- The reel configuration is stored in a file "reelconfig.info" that the EGM system reads in at startup. You can assume it was created in whatever format you want.
- The paytable is stored in a file named "paytable.info" that the EGM system reads in at startup. You can assume it was created in whatever format you want.
- In order to comply with the Nevada State Gaming Control Board regulations, the information that needs to be available for auditing purpose has to be stored on non-volatile storage. (For this design you do not have to store the information in a fault-tolerant way, although technically this would also be required by the Nevada State Gaming Control Board).

Task 1: Interaction Model

Establish an *Interaction Model* for the following system operations of the *Gamble* use case: `setBet`, `setPaylines`, `startPlay`, `pJackpotWonValue`, `handpayCompleted` and `resetToCredits`. Additionally, you must do a design for that part of the `audit` system operation that is described in the operation model. Finally, please design an operation `systemStartup` that correctly initialies any data structures that your system needs. Note that:

- You can either submit communication diagrams or sequence diagrams for your interaction models.

- The interaction model only focusses on method invocations between objects. If in your design there is a method that does some complex calculation (such as, for example, triangularizing a matrix) you do not need to show this in your diagrams. However, it could be nice to provide a short textual description that outlines what this method does.

- To keep the size of the design reasonable, you do not have to design any of the other operations that are part of the *Gamble* use case. You can simply assume that these operations use your design data structures in a way that is convenient and consistent with the operations you are designing. If your designs are depending on some other operation to have done something previously that is related to your specific design (and hence not specified in the operation schemas provided in section 6), then please state that assumption in a comment.

- You do not have to design in detail how output communication to actors is handled. It is enough to introduce a dedicated design class that takes care of delivering messages to an actor. For example, to communicate with the coin hopper, a "CoinHopper" class can be introduced into the design, with the method `void ejectCoins(int amount)`. You can then assume that some hardware specialist will design some coin hopper device driver that will provide that functionality.

- You do not have to design the communication with the touch screen at all. You can assume that a separate design team develops "View" classes that are attached to the system's state using the *Observer* design pattern. Thanks to the efforts of this team, all changes to the system's state that need to be communicated to the player on the touch screen are taken care of. This means that, for example, you do not have to include the currentCredits communication message that needs to be sent to the touch screen in your design. It suffices to assign a new value to the attribute that you are using to store the current number of credits of the player.

- You do not have to design in detail how input communication from actors is handled. You can simply assume that for each actor there is some object in the design that knows how to listen to incoming messages from that actor, and that knows how to decode the parameters of the message, if any. Once this is done, this object invokes the controller object (of your choice). In other words: please start each interaction model with a message coming out of nowhere and ending up at your chosen controller object with the parameters initialized correctly.

Task 2: Design Class Model

Establish a *Design Class Model* that shows the (private) **attributes** and **methods** (public and private) of all design classes used in task 1, as well as their **navigable associations** and **inheritance** relationships. Note that:

- You do not have to show usage dependencies.

- All attributes of classes are to be made private. If their values are to be available to other classes, then appropriate *getter* or *setter* methods have to be defined.

8 Hand-In

Please hand in a paper or pdf copy of your solution until Tuesday November 19th. You can hand in electronically by sending an email to Omar.Alam@mail.mcgill.ca with the title "COMP-533 Final of yournames" and cc me as well (Joerg.Kienzle@mcgill.ca). If you don't get an acknowledgment for your email, send us another email (without attachment, but putting the handin somewhere where we can download it).

Remember that you are allowed to work in groups of 2, but not with a person you worked with for a previous assignment. Actually, since elaborating a design is a very creative activity, we really encourage you to work in teams. Please hand in a single copy with both names on it.