Arcade Game Maker Product Line – Requirements Model

ArcadeGame Team

July 2003

Table of Contents

Revision Control Table				
Version Number	**Date Revised**	**Revision Type A-Add, D-Delete, M-Modify**	**Description of Change**	**Person Responsible**
2.0	6/1/04	M	Responded to review comments	JDMcGregor

List of Figures

Overview

This is the requirements document for the Arcade Game Maker product line. Its purpose is to provide the specifications for the products that will be built as part of the product line.

1.1 Identification

The Arcade Game Maker Product Line will produce a series of arcade games. Each game is a one or two player game in which the players control, to some degree, the moving objects. The objective is to score points. How points are scored varies from one game to another. The games allow player-controlled interaction times.

1.2 Document Map

The Arcade Game Maker Product Line is described in a series of documents. These documents depend on each other as shown in Figure 1. These dependencies show a possible order in which the documents could be read for the first time. Once the reader is familiar with the documents, the reader can go directly to the document needed.

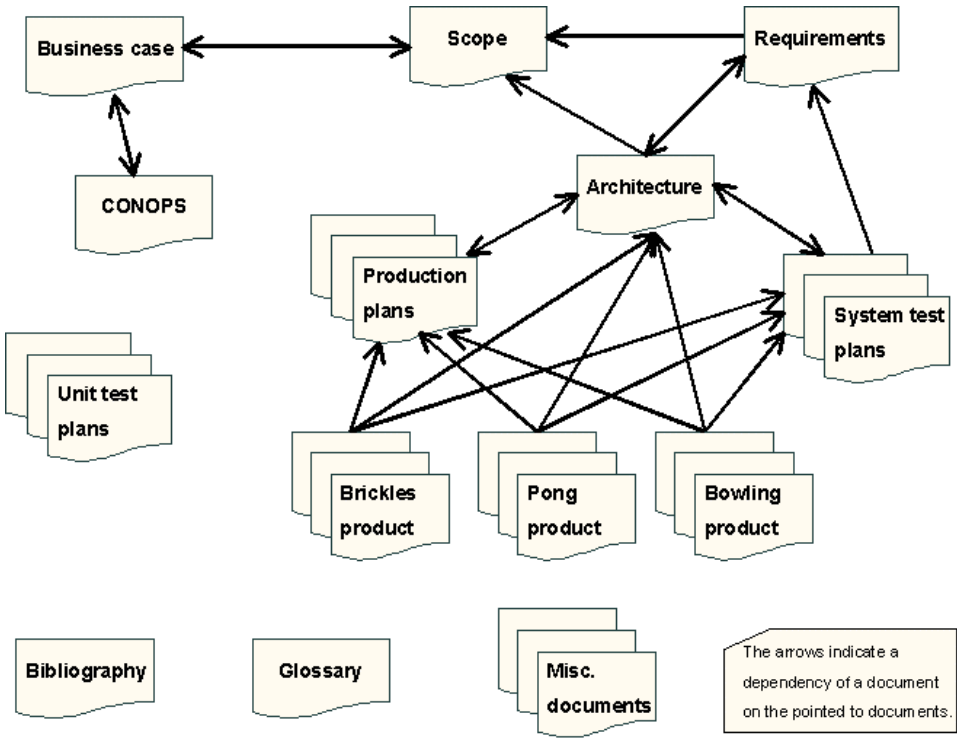

Figure 1- Document Map

1.3 Concepts

Refer to the Glossary document for definitions of basic concepts.

1.4 Reusable Components

This document establishes the high-level context for work in the product line. In a product line, components are designed to be reusable within the context of the product line. That is, no attempt is made to make a component “as general as possible”. Each design decision is made with regard to the extent of the products in the product line. The architecture further refines the context defined in this document.

1.5 Readership

This document is intended to provide some level of information to all of the stakeholders in the Arcade Game Maker product line. Managers will find the information needed to support product planning. Product line analysts will find the information necessary to support commonality and variability analysis. Product developers will find the rationale for each product’s membership in the product line.

2 Use Case Model

The use case diagram in Figure 2 provides an overview of the use cases for the Arcade Game Maker product line.

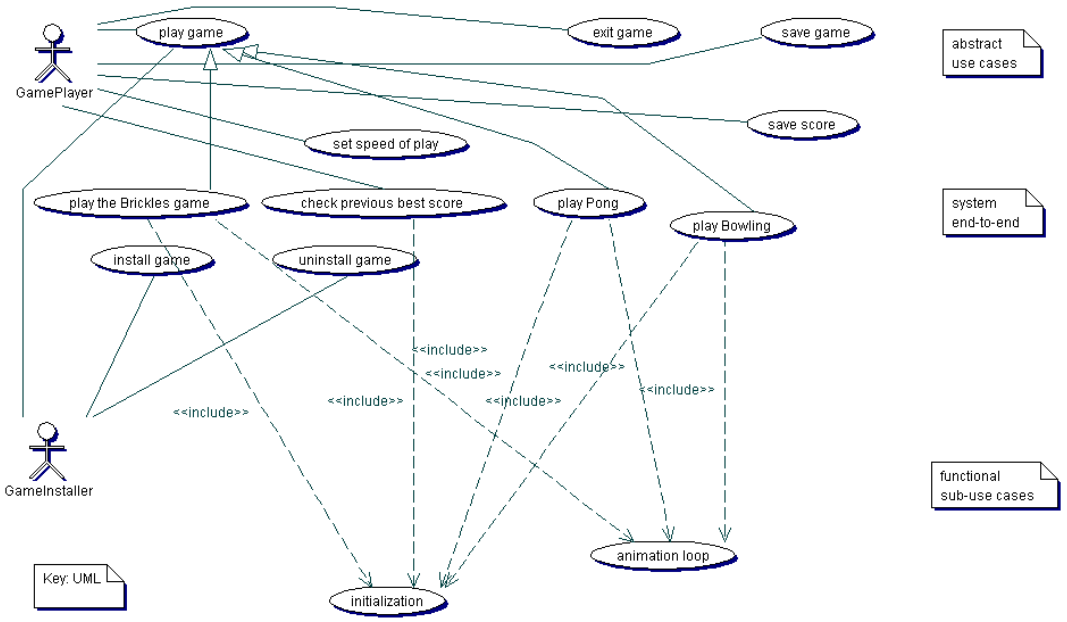

Figure 2 - Use Case Diagram

Table 1 – Use Cases

AGM001 – Play the Game
AGM002 – Exit the Game
AGM003 – Change Case – Save the Game
AGM004 – Change Case – Save Score

AGM005 –Change Case – Check Previous Best Score
AGM006 – Play Brickles
AGM007 – Play Pong
AGM008 – Play Bowling
AGM009 – Initialization
AGM010 –Animation Loop
AGM011 – Install Game
AGM012 – Uninstall Game
AGM013 – Set the Speed of Play

3 Domain Model

Prior to identifying specific requirements, a domain analysis was conducted to identify the essential concepts. These concepts form the basic vocabulary that is used in describing the use cases and the commonality and variability analysis.

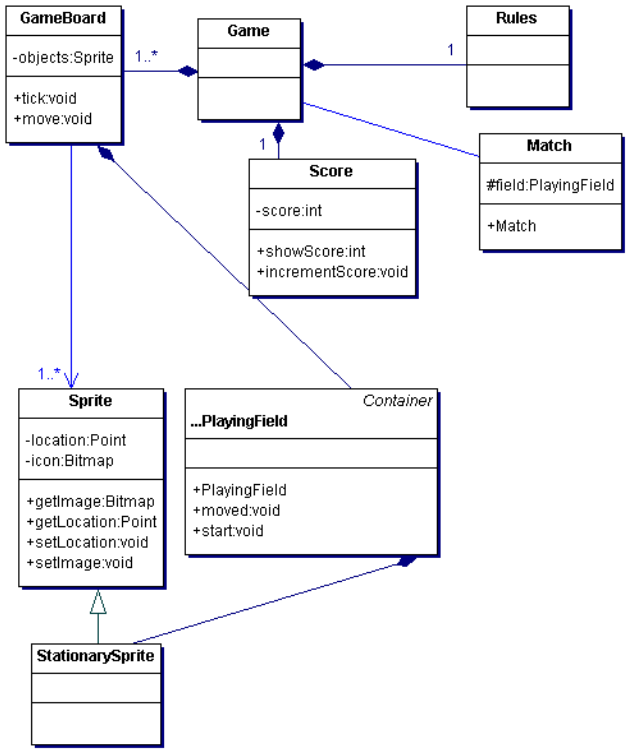

Figure 3 - Domain Model in UML

4 Commonality Analysis

4.1 Overview

In a product line, the analysis covers a set of products rather than a single one. In this section we document what is common among the products identified as belonging to the product line.

4.2 Definitions

Sprite – From some of the earliest days of computer-based games, the elements that the player could see and interact with on the screen have been referred to as Sprites.

Rule – Game play and operation is governed by rules. For example, a game may have a rule that a moving sprite that strikes a stationary sprite obeys the laws of classical mechanics. Games also have rules that define how to score in the game.

4.3 Commonalities

- Every game will have a set of Sprites
- Every game has a set of rules.
- All of the games involve motion.

4.4 Variabilities

The single biggest variation among the products is a difference in rules. Some of the rules relate to basic physical laws such as gravity or elastic collisions. These rules may be applicable to multiple games. Other rules relate to the specifics of a game. These rules can be used in all implementations of that game but don't apply to other games.

A second variation is the means by which motion is initiated. In some games the motion is inherent in the operation of the game. The action happens periodically and is driven by time. In other games the player initiates movement. The action happens whenever the player selects an action. The action is driven by the player's actions.

4.5 Parameters of Variation

Parameters are used at several levels of definition to provide the maximum variation among products. The set of Sprites that make up a game can be varied to enhance or change the game. For example, in the Brickles game, different types of bricks can be defined that exhibit different types of behavior during a collision.

4.6 Issues

- What is the scope within which a variation is constant?

- Should parameterized units be saved as assets or should they always be built up for each product?

4.7 Scenarios

- The developer is assigned to build a new product that incorporates a game that has been implemented before. There will be many assets available.

- The developer is assigned to build a new product that incorporates a game that has not been implemented before. Some assets are available because some of the physical rules are the same as in other games that have been implemented. Scoring rules will have to be constructed from scratch.

5 Feature Model

In this section we present the results of the FODA for the arcade game domain.

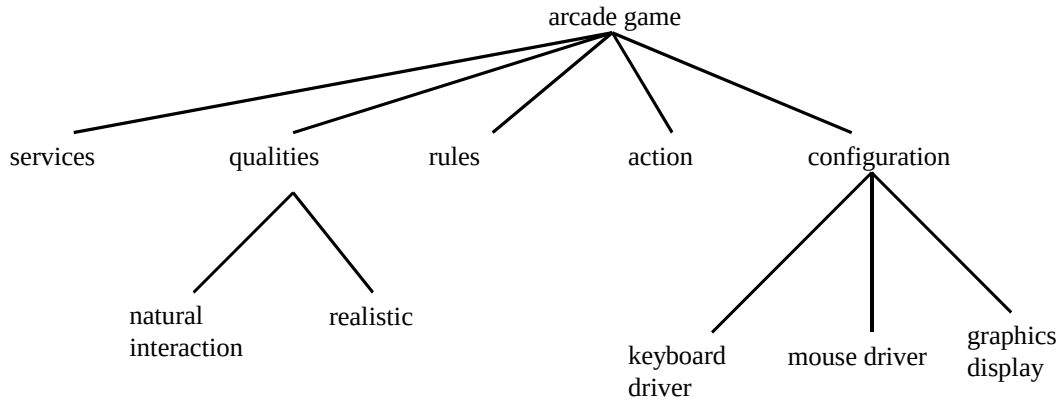

Figure 4 - Top level of feature analysis

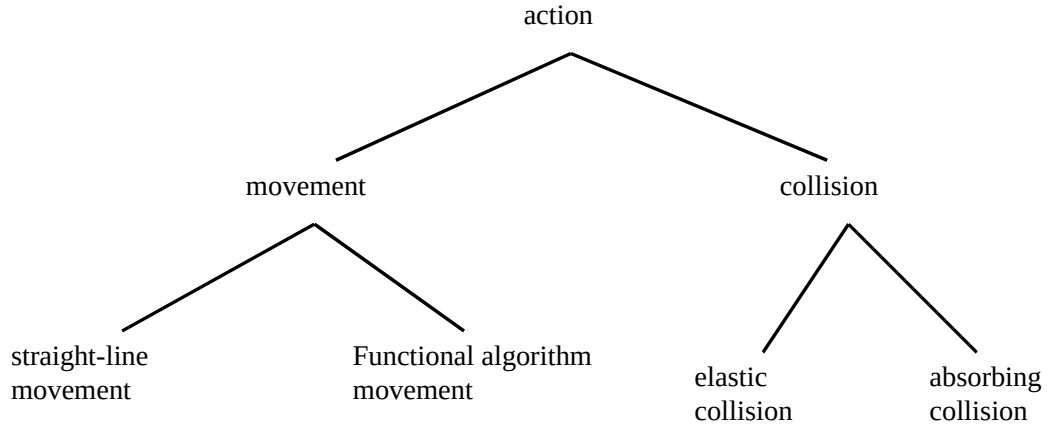

Figure 5 - Continuation of feature analysis

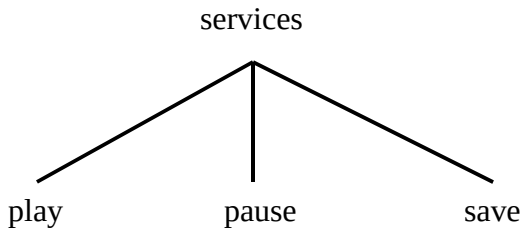

Figure 6 - feature analysis of the services feature

6 Non-functional requirements

In this section we will record those requirements that do not lend themselves to use cases.

6.1 System operational requirements

6.1.1 Performance

The action of the game must be sufficiently fast to seem continuous to the user. There should be no blurring of the graphics. Research in human computer interface has shown that this requires a refresh every fifty milli-seconds for the motion in the games to appear smooth.

6.1.2 Display quality

The colors chosen for the games must be such that the color of one element does not impair the user's view of another element.

6.2 System development requirements

6.2.1 Evolvability

The assets used in the first increment will be the basis for the products in the next two increments. Assets of the first increment must be capable of being modified to suit the second increment by a single programmer in less than a week. The architecture must be capable of being modified within two weeks.

6.2.2 Maintainability

Some of the core assets will "age" over time as new releases of the environment are put into play. These assets must be maintained current with the components with which they interact. These assets must be capable of being integrated with a new release of the environment in three days by a single experienced systems programmer.

7 Attached Processes[1]

7.1 Building the Requirements Model

AGM will use the requirements modeling technique defined in [Chastek 01]. We will not repeat that information here. Figure 7 shows the basic process. Since this is for a product line, each activity refers to all of the products in the product line.

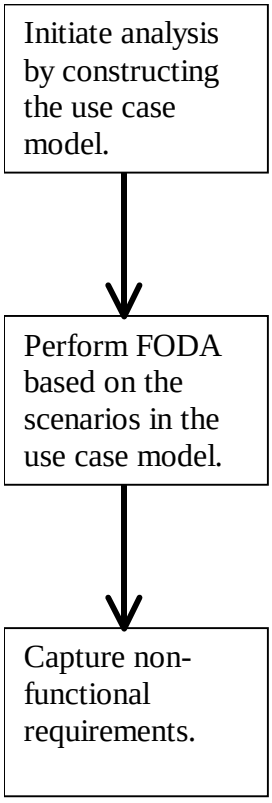

Figure 7 - Building Requirements Model

7.2 Modifying the Requirements Model

The requirements model will evolve over time as new requirements are added, some may be deleted, some may change from a change case to a current requirement, and some may be

[1] This section is the "attached process" described in [Clements 02]. For the product line requirements model this process defines how the requirements model is built initially. The process focuses mainly on modifying the requirements model of the product line.

modified to change the scope of the requirement. Each time a requirement is added or deleted, the person making the change must check the model for consistency, correctness and completeness. Figure 8 illustrates this process.

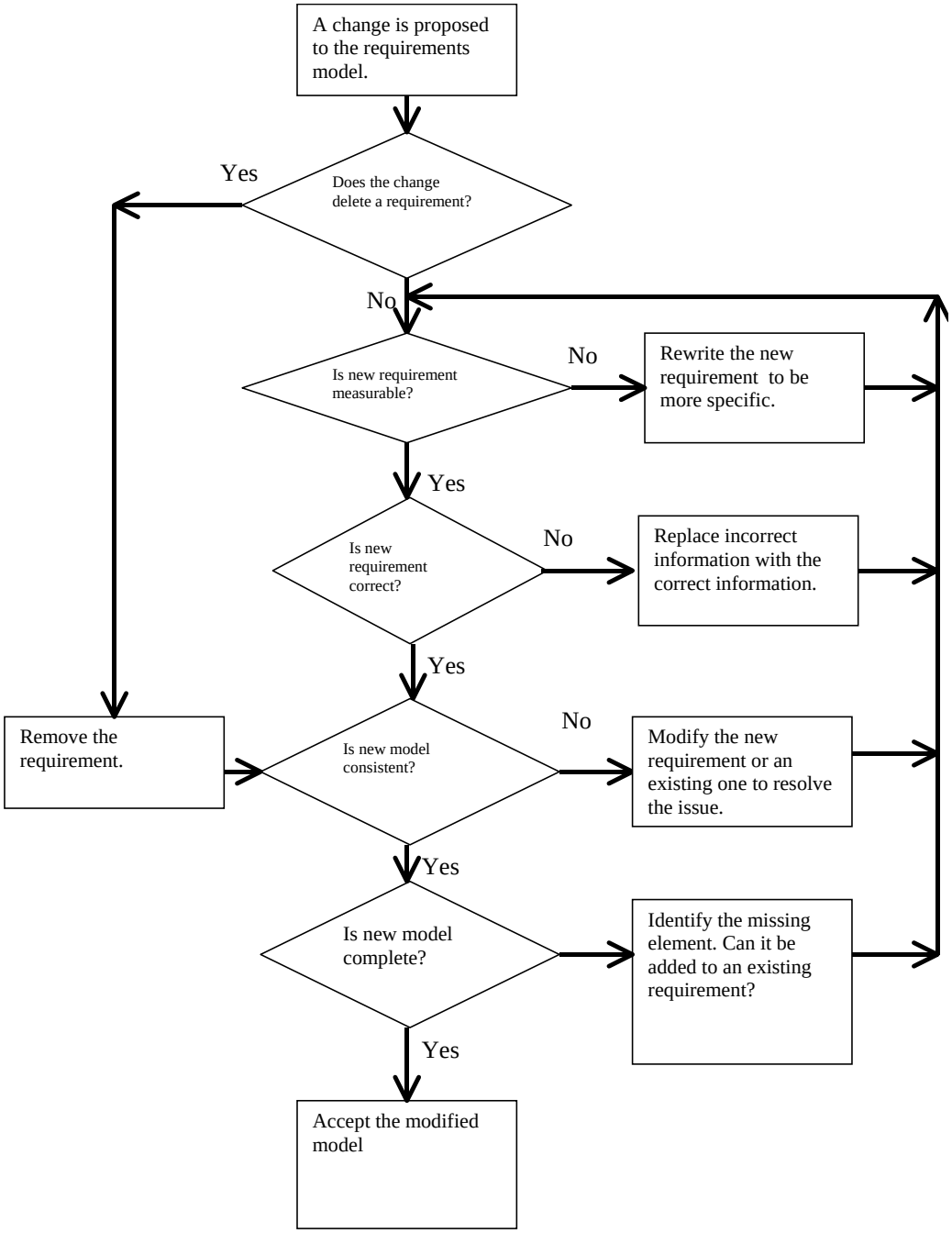

Figure 8 - Process flow for adding or deleting a requirement

8 References and Further Reading

For references see the Bibliography document.

Appendix A – Use Cases

Play the Game

Use Case ID: AGM001

Use Case Level: **abstract**

Scenario

Actor: GamePlayer or GameInstaller

Pre-Conditions: AGM011 has completed successfully

Detailed Description

Trigger: Actor selects game executable and initiates execution

Actor	System responds by
Selects **`PLAY`** from the menu	Initializes the game and displays the gameboard
Left mouse click (or equivalent) to begin play	Starting game action
Uses left mouse button (or equivalent) or keyboard to enter commands	Responds to the command in the expected manner
Responds to Won/Lost/Tied dialog with left mouse click (or equivalent)	Returns the gameboard to its initialized, ready to play state

Post-conditions: Actor has Won/Lost/Tied and the game is ready to play again

Alternative Courses of Action:

Actor	System responds by
At any time the actor may select **`EXIT`** from the menu	See use case AGM002

Extensions:

Actor	System responds by
See use case AGM006	
See use case AGM007	
See use case AGM008	

Exceptions:

Actor	System responds by

Concurrent Uses:

Related Use Cases: Exit the game

External Supporting Information

Requirement Originator: domain analyst

Rationale For Requirement: This is the main purpose of the product

Additional Relevant Requirements:

Decision Support

Frequency: on demand

Criticality: high

Risk: low

Modification History

Use Case Recorder: John D. McGregor

Initiation Date: Tuesday, June 01, 2004, at 4:52 PM

Last Modified: Tuesday, June 01, 2004, at 4:52 PM by John McGregor

Exit the Game

Use Case ID: AGM002

Use Case Level: **System End-to-End**

Scenario

Actor: GamePlayer or GameInstaller

Pre-Conditions: Game is running

Detailed Description

Trigger:

Actor	System responds by
Selects **EXIT** from menu	Asking if the actor would like to save the game or cancel the action
If actor saves game	Saving and exiting the game

Post-conditions: game is terminated

Alternative Courses of Action:

Actor	System responds by
If the actor cancels the **EXIT** action	Returning to the action where it was when the selection was made

Actor	System responds by
may initiate **EXIT** by performing a left button click (or equivalent) on the upper right hand corner of the game window	Asking if the actor would like to save the game or cancel the action
If actor saves game	Saving and exiting the game

Extensions:

Actor	System responds by

Exceptions:

Actor	System responds by

Concurrent Uses:

Related Use Cases: AGM001 – play the game

External Supporting Information

Requirement Originator: product planning

Rationale For Requirement: usual action for an interactive game

Additional Relevant Requirements:

Decision Support

Frequency: low – only once per game startup

Criticality: low

Risk: low

Modification History

Use Case Recorder: John D. McGregor

Initiation Date: Tuesday, June 01, 2004, at 4:52 PM

Last Modified: Tuesday, June 01, 2004, at 4:52 PM by John McGregor

Change Case - Save the Game

Use Case ID: AGM003

Use Case Level: **System end-to-end**

Scenario

Actor: GamePlayer or GameInstaller

Pre-Conditions: game executable is running and a game has been started

Detailed Description

Trigger:

Actor	System responds by
Selects the **`SAVE`** option in the	Allowing the actor to specify a file

menu	name
	Writing game data to the file
	Returning to the game in the status as before the save

Post-conditions: the current state of the game has been written to the specified file OR the action has been cancelled

Alternative Courses of Action:

Actor	System responds by
Selects the **`EXIT`** menu option	See use case AGM002

Extensions:

Actor	System responds by

Exceptions:

Actor	System responds by
Selects the **`SAVE`** option in the menu	Allowing the actor to specify a file name
	Raising an exception because the disk is full
Selects a different disk	System attempts to save again

	System identifies an existing file with the same name as specified in the Save dialog.
	Raise ExistingFileException
Choosing a different name	System writes the file
OR	
Agrees to overwrite the existing file	System writes the file

Concurrent Uses:

Related Use Cases: AGM002 – Exit the Game

External Supporting Information

Requirement Originator: product planning

Rationale For Requirement: a convenience feature for users

Additional Relevant Requirements:

Decision Support

Frequency: on demand

Criticality: medium

Risk: medium – other files might be corrupted

Modification History

Use Case Recorder: John D. McGregor

Initiation Date: Tuesday, June 01, 2004, at 4:52 PM

Last Modified: Tuesday, June 01, 2004, at 4:52 PM by John McGregor

Change Case - Save Score

Use Case ID: AGM004

Use Case Level: **System End-to-End**

Scenario

Actor: GamePlayer

Pre-Conditions: game is running and a game is in progress

Detailed Description

Trigger:

Actor	System responds by
Selects **`SAVE SCORE`** from menu	Allowing the actor to specify a file name
	If the file does not exist, create new file
	Write score to the file
	Returning to the game in the status as before the save

Post-conditions: file has been written OR action has been cancelled

Alternative Courses of Action:

Actor	System responds by
Selects **`SAVE SCORE`** from menu	Allowing the actor to specify a file name
	If the file exists, overwrite existing score Else create new file and write score

Extensions:

Actor	System responds by

Exceptions:

Actor	System responds by
Selects the **`SAVE SCORE`** option in the menu	Allowing the actor to specify a file name
	Raising Exception because the disk is full
Selects a different disk, if available	System attempts to save again

Concurrent Uses:

Related Use Cases: AGM005 Check Previous Best Score

External Supporting Information

Requirement Originator: John D. McGregor

Rationale For Requirement: Want to provide the user with motivation

Additional Relevant Requirements:

Decision Support

Frequency: on-demand

Criticality: medium

Risk: low

Modification History

Use Case Recorder: John D. McGregor

Initiation Date: Tuesday, June 01, 2004, at 4:52 PM

Last Modified: Tuesday, June 01, 2004, at 4:52 PM by John McGregor

Change Case - Check Previous Best Score

Use Case ID: AGM005

Use Case Level: **System End-to-End**

Scenario

Actor: GamePlayer

Pre-Conditions: game is running

Detailed Description

Trigger:

Actor	System responds by
Selects **`CHECK PREVIOUS BEST SCORE`**	Allowing the actor to specify a file name
	Reads the file and returns score in a dialog box
Selects OK on dialog to continue	Returns to state before select

Post-conditions: stored score has been shown to actor

Alternative Courses of Action:

Actor	System responds by

Extensions:

Actor	System responds by

Exceptions:

Actor	System responds by
Selects **CHECK PREVIOUS BEST SCORE**	Allowing the actor to specify a file name
	Finds that file does not exist
Selects **OK** on dialog to continue	Returns to state before select

Concurrent Uses:

Related Use Cases: AGM004 – Save Score

External Supporting Information

Requirement Originator: product planning

Rationale For Requirement: a motivating feature

Additional Relevant Requirements:

Decision Support

Frequency: on demand

Criticality: medium

Risk: low

Modification History

Use Case Recorder: John D. McGregor

Initiation Date: Tuesday, June 01, 2004, at 4:52 PM

Last Modified: Tuesday, June 01, 2004, at 4:52 PM by John McGregor

Systems Staff

Play Brickles

Use Case ID: AGM006

Use Case Level: **System End-to-End**

Scenario

Actor: GamePlayer and GameInstaller

Pre-Conditions: AGM011 has completed successfully

Detailed Description

Trigger:

Actor	System responds by
Selects **`PLAY`** from the menu	Initializes the game and displays the gameboard
Left mouse click (or equivalent) to begin play	Starting game action
Uses left mouse button (or equiva-	Moving the Paddle horizontally to

lent) or keyboard to enter commands	follow the pointing device track
	After each movement of the Puck, checking for a collision with another object If Puck collides with the ceiling, it is reflected back into the playing area If the Puck collides with a wall, the Puck is reflected back into the playing area If the Puck collides with the Floor, the Puck ceases to exist. A new Puck is requested and provided if the maximum number of Pucks has not been reached. If the maximum has been reached then the LOST dialog is presented. If the Puck collides with a Brick, the action is defined by the type of Brick. When the Puck collides with the last Brick, the WON dialog is presented.
Responds to Won/Lost dialog with left mouse click (or equivalent)	Returns the gameboard to its initialized, ready to play state

Post-conditions: game has been played

Alternative Courses of Action:

Actor	System responds by

Extensions:

Actor	System responds by

Exceptions:

Actor	System responds by

Concurrent Uses:

Related Use Cases:

External Supporting Information

Requirement Originator: product planning

Rationale For Requirement: this is the main action for one of the products

Additional Relevant Requirements:

Decision Support

Frequency: on demand

Criticality: high

Risk: low

Modification History

Use Case Recorder: John D. McGregor

Initiation Date: Tuesday, June 01, 2004, at 4:52 PM

Last Modified: Tuesday, June 01, 2004, at 4:52 PM by John McGregor

Play Pong

Use Case ID: AGM007

Use Case Level: **System End-to-End**

Scenario

Actor: GamePlayer and GameInstaller

Pre-Conditions: AGM011 has completed successfully

Detailed Description

Trigger:

Actor	System responds by
Selects **`PLAY`** from the menu	Initializes the game and displays the gameboard

Left mouse click (or equivalent) to begin play	Starting game action

Post-conditions: game has been played

Alternative Courses of Action:

Actor	System responds by

Extensions:

Actor	System responds by

Exceptions:

Actor	System responds by

Concurrent Uses:

Related Use Cases:

External Supporting Information

Requirement Originator: product planner

Rationale For Requirement: it is the main action of one of the products

Additional Relevant Requirements:

Decision Support

Frequency: on demand

Criticality: high

Risk: low

Modification History

Use Case Recorder: John D. McGregor

Initiation Date: Tuesday, June 01, 2004, at 4:52 PM

Last Modified: Tuesday, June 01, 2004, at 4:52 PM by John McGregor

Play Bowling

Use Case ID: AGM008

Use Case Level: **System End-to-End**

Scenario

Actor: GamePlayer and GameInstaller

Pre-Conditions: AGM011 has completed successfully

Detailed Description

Trigger:

Actor	System responds by
Selects **`PLAY`** from the menu	Initializes the game and displays the gameboard
Left mouse click (or equivalent) to begin play	Starting game action
Repeat the following for 10 frames plus a bonus throw	
Positions the mouse and left clicks (or equivalent) to send ball down alley	Moving the ball down the alley using a randomly selected algorithm. When the ball reaches the pins, any collisions result in the pins moving in ways determined by the physics of the collision.
	Counting number of pins knocked down
Positions the mouse and left clicks (or equivalent) to send ball down alley	Moving the ball down the alley using a randomly selected algorithm. When the ball reaches the pins, any collisions result in the pins moving in ways determined by the physics of the collision.
	Score is computed

Post-conditions: game has been played

Alternative Courses of Action:

Actor	System responds by

Extensions:

Actor	System responds by

Exceptions:

Actor	System responds by

Concurrent Uses:

Related Use Cases:

External Supporting Information

Requirement Originator: product planning

Rationale For Requirement: this is the main action of one of the products

Additional Relevant Requirements:

Decision Support

Frequency: on demand

Criticality: high

Risk: low

Modification History

Use Case Recorder: John D. McGregor

Initiation Date: Tuesday, June 01, 2004, at 4:52 PM

Last Modified: Tuesday, June 01, 2004, at 4:52 PM by John McGregor

Initialization

Use Case ID: AGM009

Use Case Level: **function sub-use case**

Scenario

Actor: Game

Pre-Conditions: AGM006, AGM007, or AGM008 has begun operation

Detailed Description

Trigger:

Actor	System responds by
	Creating the standard instances of the required classes
	Entering the READY state

Post-conditions: game is ready to operate

Alternative Courses of Action:

Actor	System responds by

Extensions:

Actor	System responds by
Selects **`LOAD GAME`** from the menu	Presenting a file chooser box
	Opening the file that is indicated
	Reading and constructing the game objects

Exceptions:

Actor	System responds by
	Running out of memory while creating objects
	Displaying the ERROR dialog
	Destroying objects already created

Concurrent Uses:

Related Use Cases:

External Supporting Information

Requirement Originator: system engineer

Rationale For Requirement: a natural module within the program

Additional Relevant Requirements:

Decision Support

Frequency: once per game startup

Criticality: high

Risk: medium

Modification History

Use Case Recorder: John D. McGregor

Initiation Date: Tuesday, June 01, 2004, at 4:52 PM

Last Modified: Tuesday, June 01, 2004, at 4:52 PM by John McGregor

Animation loop

Use Case ID: AGM010

Use Case Level: **functional sub-use case**

Scenario

Actor: game

Pre-Conditions: AGM009 has operated successfully and the user has done a left click on the mouse (or equivalent)

Detailed Description

Trigger:

Actor	System responds by
	Generating periodic signals and sending them to the game
	Moving all objects one step according to their movement algorithm
	Checking for collisions and executing the collision algorithms of the objects

Post-conditions: the game is completed

Alternative Courses of Action:

Actor	System responds by
Presses the left mouse button (or equivalent)	Pausing the movement of the game

Extensions:

Actor	System responds by

Exceptions:

Actor	System responds by

Concurrent Uses:

Related Use Cases:

External Supporting Information

Requirement Originator: system engineer

Rationale For Requirement: the main action sequence in a game is standard in all products

Additional Relevant Requirements:

Decision Support

Frequency: once for every playing of the game

Criticality: high

Risk: medium

Modification History

Use Case Recorder: John D. McGregor

Initiation Date: Tuesday, June 01, 2004, at 4:52 PM

Last Modified: Tuesday, June 01, 2004, at 4:52 PM by John McGregor

Install Game

Use Case ID: AGM011

Use Case Level: **System end-to-end**

Scenario

Actor: GameInstaller

Pre-Conditions: none

Detailed Description

Trigger:

Actor	System responds by
Selects the installer executable to execute	Presenting a file chooser to allow selection of a directory in which to place the game
Selects a directory	Places game files in the directory

Post-conditions: game is installed

Alternative Courses of Action:

Actor	System responds by

Extensions:

Actor	System responds by

Exceptions:

Actor	System responds by
	Finding insufficient space to which to write files
	Displaying the OUT OF SPACE dialog
Clicks left mouse button (or equivalent) on OK button	Exiting the game

Concurrent Uses:

Related Use Cases: AGM012 – Uninstall Game

External Supporting Information

Requirement Originator: product planning

Rationale For Requirement: necessary to get game operational

Additional Relevant Requirements:

Decision Support

Frequency: very seldom

Criticality: high

Risk: low

Modification History

Use Case Recorder: John D. McGregor

Initiation Date: Tuesday, June 01, 2004, at 4:52 PM

Last Modified: Tuesday, June 01, 2004, at 4:52 PM by John McGregor

Uninstall Game

Use Case ID: AGM012

Use Case Level: **System End-to-End**

Scenario

Actor: GameInstaller

Pre-Conditions: AGM011 completed successfully

Detailed Description

Trigger:

Actor	System responds by
Selects **`UNINSTALL`** from the menu	Presenting a file chooser to the actor
Selects directory where game is stored	Erases files in the directory
	Presents the UNISTALL COMPLETED dialog
Selects the OK button in the dialog box	

Post-conditions: all disk space taken up by the game is reclaimed

Alternative Courses of Action:

Actor	System responds by

Extensions:

Actor	System responds by

Exceptions:

Actor	System responds by

Concurrent Uses:

Related Use Cases: AGM011 – Install game

External Supporting Information

Requirement Originator: product planning

Rational For Requirement: a feature of the product

Additional Relevant Requirements:

Decision Support

Frequency: very seldom

Criticality: low – can be done manually

Risk: medium – might erase the wrong files

Modification History

Use Case Recorder: John D. McGregor

Initiation Date: Tuesday, June 01, 2004, at 4:52 PM

Last Modified: Tuesday, June 01, 2004, at 4:52 PM by John McGregor

Set Speed of Play

Use Case ID: AGM013

Use Case Level: **System End-to-End**

Scenario

Actor: GamePlayer

Pre-Conditions: game is executing and game has a user interface control that allows selection of play speed

Detailed Description

Trigger:

Actor	System responds by
Uses control on the user interface and indicates a desired speed	Changing the speed to match the player's selection

Post-conditions: game is operating at the selected speed

Alternative Courses of Action:

Actor	System responds by

Extensions:

Actor	System responds by

Exceptions:

Actor	System responds by

Concurrent Uses: game may be executing while this selection is made

Related Use Cases:

External Supporting Information

Requirement Originator: product planning

Rational For Requirement: a feature of the product

Additional Relevant Requirements:

Decision Support

Frequency: very seldom

Criticality: low

Risk: low

Modification History

Use Case Recorder: John D. McGregor

Initiation Date: Tuesday, June 01, 2004, at 4:52 PM

Last Modified: Tuesday, June 01, 2004, at 4:52 PM by John McGregor

Appendix B - Actor Definitions

Actor Profile - GamePlayer

Actor Name: GamePlayer Abstract: Yes ___ No √

Description: This actor is the usual user of the system.

Skill Level: medium

Use Case ID	Primary/Secondary	Personality[2]	Relative Frequency
AGM002	Secondary	Initiator	low
AGM003	Secondary	Initiator	medium
AGM004	Secondary	Initiator	low
AGM005	Secondary	Initiator	low
AGM006	Secondary	Initiator	high
AGM007	Secondary	Initiator	high
AGM008	Secondary	Initiator	high

[2] Initiator, server, receiver, facilitator

Actor Profile - GameInstaller

Actor Name: GameInstaller Abstract: Yes ___ No √

Description: This actor is the installer of the system. This actor is a very infrequent user of the system.

Skill Level: high

Use Case ID	Primary/Secondary	Personality[3]	Relative Frequency
AGM002	Secondary	Initiator	low
AGM003	Secondary	Initiator	low
AGM004	Secondary	Initiator	low
AGM005	Secondary	Initiator	low
AGM006	Secondary	Initiator	low
AGM007	Secondary	Initiator	low
AGM008	Secondary	Initiator	low
AGM011	Primary	Initiator	low
AGM012	Primary	Initiator	low

Systems Staff

[3] Initiator, server, receiver, facilitator