Contents

1. List of students

Xing Zhao

2. Game description

This text game is built based on the famous TV series "Star Trek" in 1966-1969. It is a story about Captain James T. Kirk and the crews of the Starship USS *Enterprise* explore the galaxy and defend the United Federation of Planets. The Klingons are humanoid warrior species and initially intended to be antagonists for the crew of the USS *Enterprise.* In the game, the player is taking role of Captain Kirk commands the USS *Enterprise* fleet on a mission to hunt down and destroy the invading fleet of Klingons warships in the galaxy.

The game starts with a short greeting and description on the potential random element of gaining points in the game and the rule of the game. Each game starts with 10 Klingons and 1 hidden supply position randomly spreading throughout the galaxy.

The galaxy map is arranged as an 8 by 8 gird of quadrant. The number of items or units, Klingons and supply station are fixed for each mission but their exact position is randomly generated. A new hidden supply position would be generated when *Enterprise* happened to move to the existing supply position. The supply from the hidden position is automatically loaded to *Enterprise*.

Klingon warships were represented with a " * " and the *Enterprise* itself with an "–E–". The player can use the provided menu options to move and attack. When the user initiates the attack, Klingons on the same row and column of *Enterprise* would detect the fleet and attack back. So the game requires the user to initiate attack strategically.

Movement drains the energy supply of the *Enterprise.* Be attacked by Klingons costs shields. Crash into a Klingon on moving also consumes a large amount of shields. However, all supplies are rewarded by destroying a Klingon or moving to a hidden supply position.

An operation report is displayed underneath the galaxy map after a command is issued by user.

There are three conditions or states for the *Enterprise:* GREEN, RED and DEAD. When any supply of the fleet is less than certain amount, the condition changes to RED. The condition may change from RED to GREEN when the supply increases back to a normal level. But when any supply is zero, the condition sets to DEAD (the *Enterprise* is destroyed) which is also the end of the game. "Mission failed" is displayed. The user can choose to restart a new game or quit the current game.

There is an additional menu option for emergency situations when the fleet's condition is RED, such as calling for help from the base which provides certain amounts of supplies to the *Enterprise*. However, call for help option is only offered once per mission.

The user would receive a Victory when all Klingons in the map are destroyed and a new mission would start. The *Enterprise* carries all supplies from accomplished mission to the new mission

after Victory. The user can choose to quit the game at any stage. “Mission Aborted!” is displayed upon quit is issued.

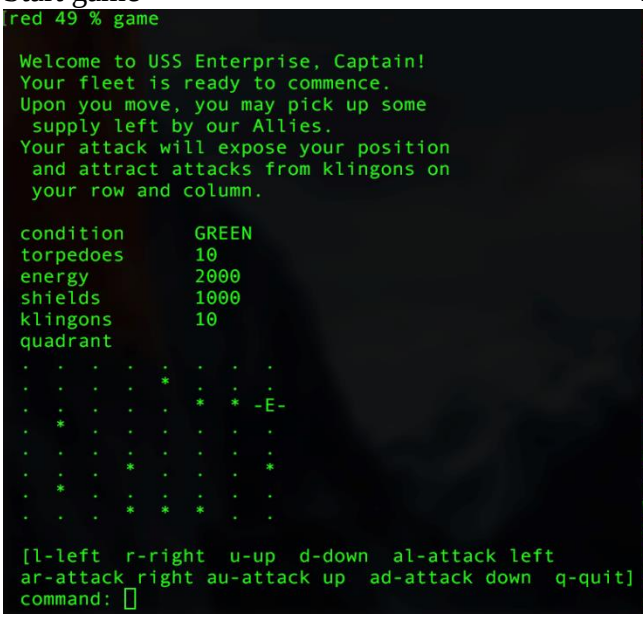
Install Star Trek

Under diectory “*star_trek*”, run “*estudio15.12 star_trek.ecf &*” and choose *compile* or *freeze*.

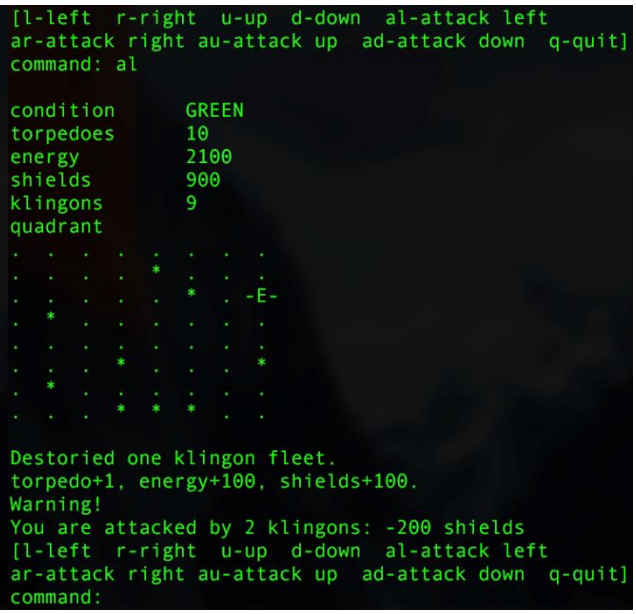
Run “*EIFGENs/star_trek/W_code/star_trek*” under directory “*star_trek*”.
Or create a symbolic link *game* “*ln –s EIFGENs/star_trek/W_code/star_trek game*” and run *game*.

Demonstration

Start game

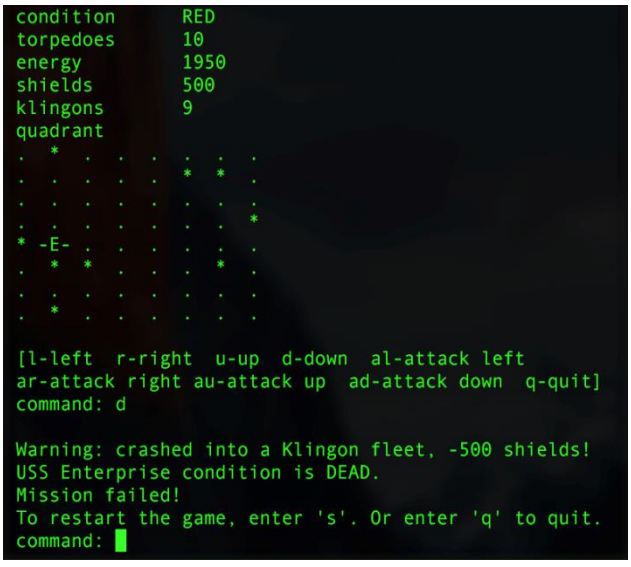


Attack left side

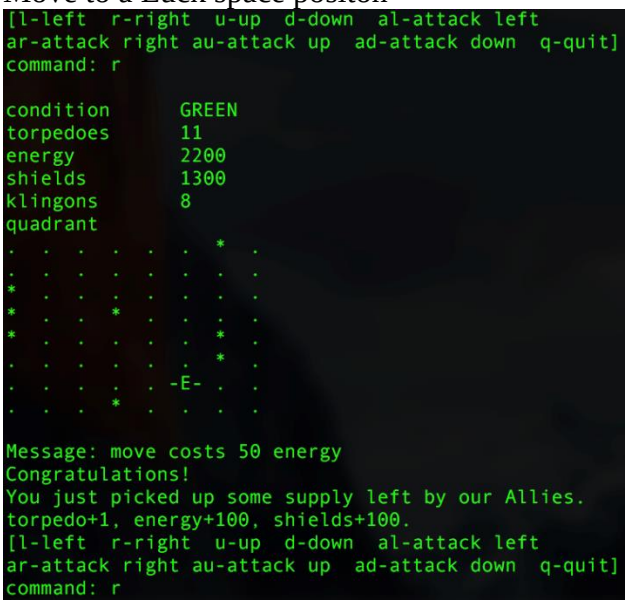


State change

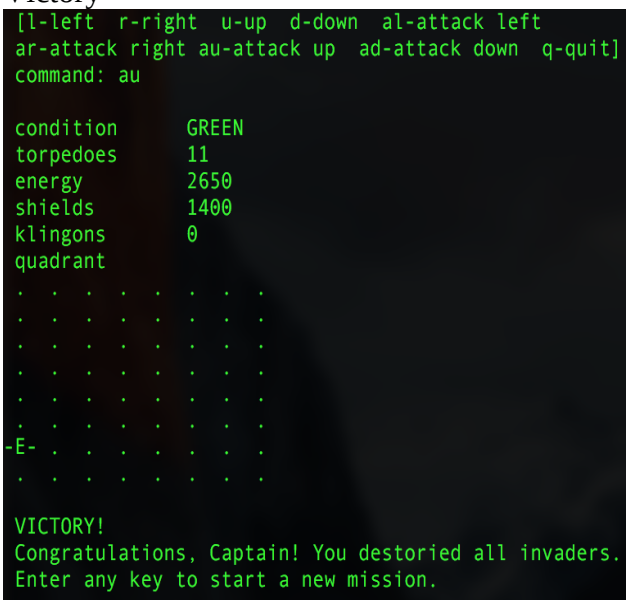
Crash and Mission Fail



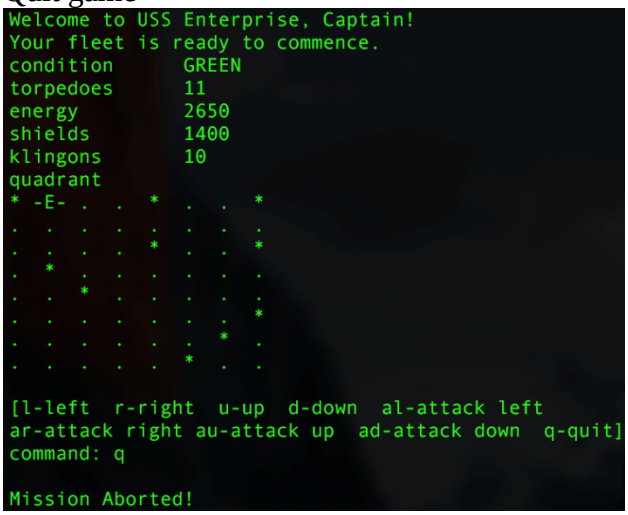
Move to a Luck space positon



Victory



Quit game



3. Brief class ADT documentation

Class **CONTRL**
Description: Controller of the game
Operations

r: RAND_SEQ
Comments: provide random features from RAND_SEQ class

start
Comments: start the game

Class Invariants
model ≠ ∅ ***and*** *view* ≠ ∅
game_units = model.game_units

Comments:
The purpose of the class invariant is to ensure the running time safety for an instance of the class

- mode and view cannot be void; otherwise the controller cannot play its role in MVC pattern and is then defect.
- game_units must always equal to the game_units of the attached model, otherwise the attached model is incorrect or there is an error during the operation between controller and model.

Class **CONTRL_ACCESS**
Description: Singleton access to CONTRL
Operations

singleton (m: SPACE_MAIN_UNITS; v: DISPLAY): CONTROL
-- create singleton access point to CONTRL
require *m* ≠ ∅ ***and*** *v* ≠ ∅
enusre *Result = Result*

Comments:
It is a singleton accessor to controller. Pre-condition requires model and view must be non-void, otherwise the class invariants of CONTRL is violated. Post-condition enforces the single instance property.

Class **DISPLAY**
Description: View of the game, display game view in text.
Operations

send (content: STRING)
-- send the content to the view or presentation
-- print out the game in this case
require *content* ≠ ∅

Comments:
Pre-condition checks that the content to be printed out is non-void since void cannot be printed out or void violates the Pre-condition of print.

receive: STRING
-- return the received content from user/input
ensure *Result = Io.last_string*
Comments:
Post-condition ensures that the returned String data is same as the user input from the command line.

Class **RAND_SEQ**
Description: Customized RANDOM, use TIME to set random seed
Operations

get_seed: INTEGER
-- return a random seed
ensure *Result = r_seed*

Comments:
Post-condition ensures that the returned random Integer is the correct random Integer generated by current instance class.

get_rnd: RANDOM
-- return a RANDOM r
ensure *Result = r*

Comments:
The Post-condition ensures that the returned RANDOM instance is the correct RANDOM instance generated by current instance class.

get_random_set (num: INTEGER; index: INTEGER): LIST[INTEGER]
-- generate an unique set of random number in range from 0 to `index'
-- the size of the set is `num'
require *num* > 0 ***and*** *index* > 0
ensure *Result.count = num*

Comments:
Pre-condition checks if the required number of random Integers is positive, otherwise it makes no sense to generate 0 or negative number of random Integer. It also checks if the max boundary of the index is positive since the index of a list starts from 1.
Post-condition ensures that the returned list contains the correct number of random Integers as the specification promises.

Class Invariants
r_time ≠ ∅ ***and*** r ≠ ∅

Comments:
The purpose of the class invariant is to ensure the running time safety for an instance of the class. TIME object and RANDOM object must not be void during the current object life cycle so that the features this class provides are valid at running time.

Class **SINGLETON**
Description: Deferred class defines Singleton allowing multiple singleton class in a system
Class Invariants
Current = the_singleton

Comments:
The invariant ensures that the current instance of this class is the only one instance in the system at running time.

Class **SPACE_MAIN_UNITS**
Description: model of the game. The main structure that contains all game units and operation features.
Operations

load_space_track
-- fill the list with SPACE_TRACKs
require *space_units* $\neq \varnothing$
ensure *count = game_units*

Comments:
Pre-condition requires that the list space_units must be non-void to execute this feature since the program depends on space_units to meet the specification. Otherwise, the program will not work. Post-condition ensures that the total game units' count is corrected assigned by checking it is same as the number of loaded units in the list.

load_klingons
-- fill the KLINGONs with random index to the list
ensure *count_klingons = max_klingons*

Comments:
Post-condition checks that the number of Klingons loaded to the list space_units is correct by ensuring it is same as max_klingons which is specified at the initiation of the current instance.

load_uss_e
-- replace USS_E randomly with a SPACE_TRACK
ensure $0 < uss\_e\_id \leq game\_units$

Comments:
Post-condition ensures that the index of *Enterprise,* uss_e_id in the list space_units is in the range of legit id.

klingons_decrease
-- USS_E destorys one klingons
require *count_klingons > 0*
ensure *count_klingons =* ***old*** *count_klingons - 1*

Comments:
Pre-condition checks the number of Klingons is positive otherwise the game should be ended with a victory if the number of Klingons is 0. Other parts of the instance must go wrong if this pre-condition is violated.
Post-condition ensures the number of Klingons decreases by 1.

hit_target (invader: INTEGER)
-- successfully destroy a Klingon at index `invader'
require *invader > 0* ***and*** *invader ≤ count*

Comments:
Pre-condition requires that the index of a Klingon must be a legit index or in the range of list space_units so that a specified Klingon can be found from the list.

attack (direction:STRING):STRING
require *direction ≠ ∅*
direction = "al" ***or*** *diection = "ar"* ***or*** *diection = "au"* ***or*** *diection = "ad"*

Comments:
Pre-condition requires that given direction must be non-void and the function or feature only accepts four options or directions to execute attack. Otherwise, it cannot process nonsense input directions.

move (direction: STRING): STRING
require *direction ≠ ∅*
direction = "l" ***or*** *direction = "r"* ***or*** *direction = "u"* ***or*** *direction = "d"*

Comments:
It is same as attack. Pre-condition requires that given direction must be non-void and the function or feature only accepts four options or directions to execute move. Otherwise, it cannot process nonsense input directions.

uss_e_swape (index: INTEGER)
-- swape USS_E unit with the unit at `index' of the list
require *0 < index ≤ count*
ensure *uss_e_id = index* ***and*** *space_units[uss_e_id].id = index*
space_units[uss_e_id].name = "-E-"

Comments:
Pre-condition requires that the index of *Enterprise* must be a legit index or in the range of list space_units so that it can be found from the list.
Post-condition ensures that uss_e_id is updated correctly and the id of *Enterprise* in the list space_units is updated correctly. Finally, it checks that if the new index of *Enterprise* has the correct type or keeps the same type in the list.

uss_e_crash (index: INTEGER)
-- USS_E crashes into the unit at `index' and replaces the unit at `index'
require $0 < index \leq count$

Comments:
Pre-condition requires that the given index must be a legit index or in the range of list space_units so that the unit of the given index is replaceable.

quadrant_out (n: INTEGER): STRING
-- quadrant information in String format. `n', size of the quadrant in n x n
require $n > 0$

Comments:
Pre-condition requires that the size of the quadrant must be positive since the size of the quadrant is at least 1 x 1.

is_victory:BOOLEAN
-- check if all kinglons are destoryed
ensure *Result = (count_klingons = 0)*

Comments:
Post-condition ensures that the victory is only true when the number of Klingons is 0. It checks the correctness of the victory specification.

scan_left (host:INTEGER; target:INTEGER):BOOLEAN
-- index `host' checks if the index `target' is on its left
-- and in the same row returns true if the target is found.
require *host > 0* ***and*** *target > 0*

Comments:
Pre-condition requires that both given values must be positive for this feature to do the comparing.

scan_right (host:INTEGER; target:INTEGER):BOOLEAN
-- index `host' checks if the index `target' is on its right
-- and in the same row returns true if the target is found.
require *host > 0* ***and*** *target > 0*

Comments:
It is same as scan_left. Pre-condition requires that both given values must be positive for this feature to do the comparing.

scan_up (host:INTEGER; target:INTEGER):BOOLEAN
-- index `host' checks if the index `target' is on its up
-- and in the same column returns true if the target is found.
require *host > 0* ***and*** *target > 0*

Comments:
Pre-condition requires that both given values must be positive for this feature to do the comparing.

scan_down (host:INTEGER; target:INTEGER):BOOLEAN
-- index `host' checks if the index `target' is on its down or bottom
-- and in the same column returns true if the target is found.
require *host > 0* ***and*** *target > 0*

Comments:
It is same as scan_down. Pre-condition requires that both given values must be positive for this feature to do the comparing.

get_x_axis (index:INTEGER; mod:INTEGER):INTEGER
-- modular calculation of x_axis (column) based on defined `mod' (number of units per line)
-- and index in the list
require *index > 0* ***and*** *mod > 0*
ensure *Result = (index - 1) \\ mod*

Comments:
Pre-condition requires that both given values must be positive for the modular calculation since index of a list starts from 1 and mod, the devisor is positive in modular calculation.
Post-condition ensures that the correctness of returning value by checking it is equal to the remainder.

get_y_axis (index:INTEGER; mod:INTEGER):INTEGER
-- modular calculation of y_axis (row) based on defined `mod' (number of units per line)
-- and index in the list
require *index > 0* ***and*** *mod > 0*
ensure *Result = (index - 1) // mod*

Comments:
Pre-condition requires that both given values must be positive for the modular calculation since index of a list starts from 1 and mod, the devisor is positive in modular calculation.
Post-condition ensures that the correctness of returning value by checking it is equal to the quotient.

Class Invariants
space_units ≠ ∅
0 ≤ *count* ≤ *game_units*
0 < *uss_e_id* ≤ *game_units*
0 ≤ *count_klingons* ≤ *max_klingons*
count = space_units.count

The purpose of the class invariant is to ensure the running time safety for each instance of the class.

- The list space_units must be non-void otherwise there must be a problem on feature "make"
- count must be non-negative and not exceed the number of the existing game units, otherwise any feature involving count should be reviewed for errors.
- The index of *Enterprise* in the list space_units must be in the range of the list.
- count should also always be synchronized with the size of the list space_units, otherwise any feature involving add or remove functions of list should be reviewed for errors.
- count_klingons, the number of Klingons in the game should always be non-negative and not exceed the defined max number. Otherwise, there must be an error on the features involving with the Klingon units.

Class **SPACE_MAIN_UNITS_ACCESS**
Description: Singleton accessor to SPACE_MAIN_UNITS
Operations

m: SPACE_MAIN_UNITS
Comments:
Create singleton access point to SPACE_MAIN_UNITS

Class Invariants
m = m

Comments:
The class invariant enforces single instance property at running time.

Class **SPACE_MAIN_VENGE_MODE**
Description: A decorator class to SPACE_MAIN_UNITS, enables venge mode: once Klingons are under the attack, each of them will scan its enemy fleet on its row and column and attack back the enemy.
Operations

venge
Comments: start venge mode

reset
-- clear history of venge mode
ensure *hit_count* = 0 ***and*** *damage_report* = “”

Comments:
Post-condition ensures that all records of the venge mode are cleared.

Class Invariants
main_units ≠ ∅

Comments:
The class invariant checks the main_units which is the game model SPACE_MAIN_UNITS should always be non-void at all life cycle of current instance since this decorator class depends on the game model.

Class **SPACE_MAIN_VENGE_MODE_ACCESS**
Description: Singleton accessor to SPACE_MAIN_VENGE_MODE instance
Operations

Singleton (m: SPACE_MAIN_UNITS): SPACE_MAIN_VENGE_MODE
-- create singleton access point to SPACE_MAIN_VENGE_MODE instance
require *m* ≠ ∅
ensure *Result = Result*

Comments:
Pre-condition requires m (game model) must be non-void; otherwise the class invariant of SPACE_MAIN_VENGE_MODE is violated. Post-condition enforces the single instance property.

Class **SPACE_UNIT**
Description: Deferred Class defines a basic abstract unit for the game
Operations

set_id (index: INTEGER)
-- set a new id `index'
require 0 < *index* ≤ *max_id*
ensure *id = index*

Comments:

Pre-condition checks if the index is in the range of legit id.
Post-condition ensures that the id is assigned correctly.

Class Invariants

$name \neq \varnothing$
$0 < id \leq max\_id$

Comments:

The purpose of the class invariant is to ensure the running time safety for an instance of the class. The identity name must always be non-void during the object life cycle and id must always be in the range as game specified.

Class **SPACE_TRACK**

Description: Inherit from SPACE_UNIT. It represents a clear galaxy space that allow fleets fly through.

Operations

set_id (index: INTEGER)
-- set a new id `index'
require $0 < index \leq max\_id$
ensure $id = index$

Comments:

Pre-condition checks if the index is in the range of legit id.
Post-condition ensures that the id is assigned correctly.

Class Invariants

name = " . "

Comments:

The purpose of the class invariant is to ensure the running time safety for an instance of the class. The identity name must always be same as " . " during the object life cycle.

Class **KLINGON**

Description: Inherit from SPACE_TRACK. It represents an invading Klingon warship.

Operations

set_id (index: INTEGER)
-- set a new id `index'
require 0 < *index* ≤ *max_id*
ensure *id* = *index*

Comments:
Pre-condition checks if the index is in the range of legit id.
Post-condition ensures that the id is assigned correctly.

Class Invariants
name = " * "

Comments:
The purpose of the class invariant is to ensure the running time safety for an instance of the class. The identity name must always be same as " * " during the object life cycle.

Class **USS_ENTERPISE**
Description: Inherit from SPACE_UNIT. USS Enterprise controlled by the player on a mission to hunt down and destroy invading Klingon warships.
Operations

set_id (index: INTEGER)
-- set a new id `index'
require 0 < *index* ≤ *max_id*
ensure *id* = *index*

Comments:
Pre-condition checks if the index is in the range of legit id.
Post-condition ensures that the id is assigned correctly.

condition_check
-- set Enterprise ship condition after every activity. e.g. move, hit, attack and get_point
require *condition* ≠ ∅

Comments:
Pre-condition requires that the condition (state) must be non-void, otherwise there is problem on condition initiation in feature "make".

move
-- Enterprise moves to a new position, consume 50 energy
ensure *energy* = ***old*** *energy* – 50

Comments:
Post-condition ensures that the energy is deducted correctly. Note there is no need to set Pre-condition since the game ends if energy reaches 0 and it can never go below 0 due to 50 can always be divided with no remainder in the game.

crash
-- Enterprise crashes into a unit, lose 500 shields
ensure *(shields =* ***old*** *shields* – 500 → ***old*** *shields* ≥ 500) ***and*** *(shields = 0 →* ***old*** *shields* ≤ 500)

Comments:
Post-condition ensures that shields is deducted correctly. Since crash causes large amount of shields lose, it is possible that there is not enough shields to be deducted by 500. Shields should always be non-negative. If the shields is less than 500, it should be set to 0 which means the *Enterprise* is DEAD or destroyed.

attack
-- Enterprise launches a torpedo
require *torpedoes* > 0
ensure *torpedoes = old torpedoes* – 1
Comments:
The pre-condition requires that there is at least 1 torpedo left to launch, otherwise this operation is invalid. The torpedoes will never reach 0, otherwise the game ends.
The post-condition ensures that torpedoes is deducted correctly.

hit
-- Enterprise is hit by Klingon, lose 100 shields
require *shields* ≥ 0
ensure *(shields =* ***old*** *shields* – 100 → ***old*** *shields* ≥ 100) ***and*** *(shields = 0 →* ***old*** *shields* ≤ 100)

Comments:
The pre-condition states that as long as shields is non-negative this feature can be executed.
The post-condition ensures the shields is deducted correctly in cases of be hit by one or multiple Klingons.

get_point
-- Enterprise destroy a Klingon, gain 1 torpedo, 100 energy, 100 shields
ensure *torpedoes =* ***old*** *torpedoes* + 1
energy = ***old*** *energy* + 100
shields = ***old*** *shields* + 100

Comments:
Post-condition ensures that rewards are added correctly.

is_alive: BOOLEAN
-- check if Enterprise is alive
ensure *Result = (condition.state ≠ "DEAD")*

Comments:
The post-condition ensures the returning result is correct by checking whether the state of *Enterprise* is not DEAD.

condition_status:STRING
-- report the state of Enterprise
ensure *Result = condition.state*

Comments:
The post-condition ensures the returning state is correct by checking the current condition.

Class Invariants
name = "-E-"
condition ≠ ∅
energy ≥ *0* ***and*** *torpedoes* ≥ *0* ***and*** *shields* ≥ *0*

Comments:
The purpose of the class invariant is to ensure the running time safety for an instance of the class.

- The identity name should always be same as "-E-".
- State is always non-void.
- All supplies are always non-negative.

Class **SPACE_UNIT_STATE**
Description: Deferred Class defines a basic abstract state class for SPACE_UNIT subclasses. It maintains the state of SPACE_UNIT if applicable.
Operations

state: STRING
Comments: provide state of the SPACE_UNIT

update (s: SPACE_UNIT)
-- update internal state based on the conditions
require *s* ≠ ∅

Comments:
Pre-condition requires the given game unit must be non_void; otherwise there is nothing to be read and updated.

Class Invariants
state ≠ ∅

Comments:
The class invariant ensures that state must be non-void at running time. The state should always be any one of the pre-defined states at running time.

Class **USS_E_STATE**
Description: Inherited from SPACE_UNIT_STATE. Maintain the state of USS Enterprise.
Operations

state: STRING
Comments: provide state of the SPACE_UNIT

update (e: SPACE_UNIT)
-- update internal state based on the conditions
require *e.name = "-E-"*
ensure *state = e.condition.state*

Comments:
Pre-condition requires the given unit type must be USS_ENTERPRISE since this class is exclusively designed to represent the state of USS_ENTERPRISE.
Post-condition ensures that the state is updated correctly by checking the unit's current state.

Class Invariants
state = "GREEN" ***or*** *state = "RED"* ***or*** *state = "DEAD"*

Comments:
The class invariant ensures that state must be any one of the pre-defined states at running time.

Class **USS_E_STATE_ACCESS**
Description: Singleton accessor to USS_E_STATE instance
Operations

uss_e_state: USS_E_STATE
Comments: create singleton access point to USS_E_STATE instance

Class Invariant
uss_e_state = uss_e_state

Comments:
The class invariant enforces single instance property at running time.

4. Contracts Documentation

Class CONTRL

```
note
	description: "Summary description for CONTRL. Controller of the game"
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class interface
	CONTRL

create
	make

feature -- public attribute

	r: RAND_SEQ
			-- provide random features from RAND_SEQ

feature -- command

	start
			-- game start

invariant
	non_void_m: model /= Void
	non_void_v: view /= Void
	game_units_correct: model.Game_units = game_units

end -- class CONTRL
```

Class CONTRL_ACCESS

```
note
	description: "Summary description for CONTRL_ACCESS.
			Singleton accessor to CONTRL instance."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

expanded class interface
	CONTRL_ACCESS

create
	default_create
```

```
feature

	Singleton (m: SPACE_MAIN_UNITS; v: DISPLAY): CONTRL
			-- create singleton access to the CONTRL instance
		require
			non_void_m: m /= Void
			non_void_v: v /= Void
		ensure
			is_singleton: Result = Result

end -- class CONTRL_ACCESS
```

Class DISPLAY

```
note
	description: "Summary description for DISPLAY. Display view in text for user."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class interface
	DISPLAY

create
	make

feature -- commands

	send (content: STRING_8)
			-- send the content to the view or presentation
			-- print out the game in this case
		require
			non_void_content: content /= Void

	receive: STRING_8
			-- return the received content from user/input
		ensure
			correct_input: Result ~ Io.last_string

end -- class DISPLAY
```

Class RAND_SQL

```
note
	description: "Summary description for RAND_SEQ. Generate random INTEGER."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class interface
	RAND_SEQ

create
	make
```

```
feature -- query

	get_seed: INTEGER_32
			-- return a random seed
		ensure
			correct_seed: Result = r_seed

	get_rnd: RANDOM
			-- return a RANDOM r
		ensure
			correct_r: Result = r

	get_random_set (num: INTEGER_32; index: INTEGER_32): LIST [INTEGER_32]
			-- generate an unique set of random number in range from 0 to
			-- `index'. The size of the set is `num'
		require
			num_positive: num > 0
			mod_positive: index > 0
		ensure
			correct_size: Result.count = num

invariant
	non_void_time: r_time /= Void
	non_void_rand: r /= Void

end -- class RAND_SEQ
```

Class MESSAGE

```
note
	description: "Summary description for MESSAGE. Game messages to notify the
	              player."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class interface
	MESSAGE

create
	default_create

feature -- Global Constants of game messages

	Greeting: STRING_8 = "%N Welcome to USS Enterprise, Captain!%N Your fleet is
	                      ready to commence.%N"
			-- message at the start of the game

	Luck_intro: STRING_8 = " Upon you move, you may pick up some%N  supply left by
	                       our Allies.%N"
			-- luck rule indroduction at the start of the game
```

```
Attack_warning: STRING_8 = " Your attack will expose your position%N  and
                           attract attacks from klingons on%N  your row and
                           column.%N%N"
            -- attack warning for venge mode

Command: STRING_8 = " command: "
            -- command

Restart: STRING_8 = " To restart the game, enter 's'. Or enter 'q' to quit.%N"
            -- restart the game

Menu: STRING_8 = " [l-left  r-right  u-up  d-down  al-attack left %N ar-attack
                  right au-attack up  ad-attack down  q-quit]%N"
            -- menu of the command options

Edge: STRING_8 = " You have reached to the edge, choice another direction.%N"
            -- reach the edge of quadrant

Point: STRING_8 = " torpedo+1, energy+100, shields+100.%N"
            -- point message

Gain_point: STRING_8 = " Destoried one klingon fleet.%N torpedo+1, energy+100,
                       shields+100.%N"
            -- destoried one klingon

Miss: STRING_8 = " No klingons on your attacked direction.%N"
            -- no target is hit

Victory: STRING_8 = " VICTORY!%N Congratulations, Captain! You destoried all
                    invaders.%N Enter any key to start a new mission.%N"
            -- victory, clear all klingons in the quadrant

Lose: STRING_8 = " USS Enterprise condition is DEAD.%N Mission failed!%N"
            -- lose the game, USS Enterprise condition is dead

Quit: STRING_8 = " Mission Aborted!%N"
            -- quite the game

Uss_e_err: STRING_8 = "ERR: USS_ENTERPRICE not loaded yet"
            -- USS_E load error

Uss_e_move_err: STRING_8 = "ERR: USS_ENTERPRICE not loaded yet"
            -- USS_E move error

Move: STRING_8 = " Message: move costs 50 energy%N"
            -- move cost message

Crash: STRING_8 = " Warning: crashed into a Klingon fleet, -500 shields!%N"
            -- crash into a klingon, lost 500 shields

Red_help: STRING_8 = " Warning!%N USS Enterprice is in RED condition.%N Enter
                     'h' to call for ammunition%N Or enter any key to pass.%N"
            -- offer hidden option when USS_E state is RED
```

```
	Red_call: STRING_8 = " Ammunition is arrived!%N torpedo+1, energy+100,
	                      shields+100.%N"
	            -- ammunition is arrived after RED state help request

	Luck: STRING_8 = " Congratulations!%N You just picked up some supply left by
	                   our Allies.%N torpedo+1, energy+100, shields+100.%N"
	            -- hit luck by moving into a lucky space position

end -- class MESSAGE
```

Class SINGLETON

```
note
	description: "Summary description for {SINGLETON}. Deferred class defines
	              singleton allowing multiple singleton class in a system"
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

deferred class interface
	SINGLETON

invariant
	only_one_instance: Current = the_singleton

end -- class SINGLETON
```

Class SPACE_MAIN_UNITS

```
note
	description: "Summary description for SPACE_MAIN_UNITS. Model of the game
	              Main body that contains all UNITS for the game and operation
	              features."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class interface
	SPACE_MAIN_UNITS

create {SPACE_MAIN_UNITS_ACCESS}
	make

feature -- model attributes

	count: INTEGER_32
	            -- total number of SPACE_UNITs

	count_klingons: INTEGER_32
	            -- number of klingons warships
```

```
		space_units: LIST [SPACE_UNIT]
				-- list contains all SPACE_UNITS

		uss_e_id: INTEGER_32
				-- index of USS_E in space_units list

feature -- global constants

		Mode: INTEGER_32 = 8
				-- mode number, game has 8 X 8 quadrant

		Game_units: INTEGER_32 = 64
				-- the total game units

		Max_klingons: INTEGER_32 = 10
				-- max number of klingons

feature -- model operations

		reset
				-- Reset model state.

		continue
				-- Start a new level game after Victory
				-- USS_E carries the same information from previous game

feature -- Report messages

		Report: MESSAGE
				-- report message, create singleton access to the MESSAGE
				 -- instance

feature -- Game units loading commands

		load_space_track
				-- fill the list with SPACE_TRACKs
			require
				non_void_list: space_units /= Void
			ensure
				list_full_loaded: count = Game_units

		load_klingons
				-- fill the KLINGONs with random index to the list
			ensure
				klingons_load_correct: count_klingons = Max_klingons

		load_uss_e
				-- replace USS_E randomly with a SPACE_TRACK
			ensure
				uss_e_id_in_range: uss_e_id > 0 and uss_e_id <= Game_units
```

```
feature -- commands

	klingons_decrease
			-- USS_E destorys one klingons
		require
			positive_count_kling: count_klingons > 0
		ensure
				count_klingons = old count_klingons - 1

	uss_e_update
			-- Perform update to the USS_E condition.

	uss_e_ammunition_call
			-- USS_E calls for more ammunition under RED state or luck point

	hit_target (invader: INTEGER_32)
			-- successfully destroy a klingon at index `invader'
		require
			invader_in_range: invader > 0 and invader <= count

	miss_target
			-- missed taget, USS_E wastes 1 torpedo

	attack (direction: STRING_8): STRING_8
			-- USS_E attacks klingon in the targated direction
		require
			a_direction_non_void: direction /= Void
			a_direction_correct: direction ~ "al" or direction ~ "ar"
					or direction ~ "au" or direction ~ "ad"

	move (direction: STRING_8): STRING_8
			-- move USS_E to the `direction' one unit position
		require
			m_direction_non_void: direction /= Void
			m_direction_correct: direction ~ "l" or direction ~ "r"
					or direction ~ "u" or direction ~ "d"

	uss_e_swape (index: INTEGER_32)
			-- swape USS_E unit with the unit at `index'
		require
			swap_index_in_range: index > 0 and index <= count
		ensure
			index_assigned: uss_e_id = index
			uss_e_id_assigned: space_units [uss_e_id].id = index
			uss_e_valid: space_units [uss_e_id].name ~ "-E-"

	uss_e_crash (index: INTEGER_32)
			-- USS_E crashes into the unit at `index' and
			-- replaces the unit at `index'
		require
			clash_index_in_range: index > 0 and index <= count
```

```
feature -- queries

	quadrant_out (n: INTEGER_32): STRING_8
			-- quadrant information in String format
			-- `n', size of the quadrant in n x n
		require
			n_positive: n > 0

	is_victory: BOOLEAN
			-- check if all kinglons are destoryed
		ensure
			victory_check: Result = (count_klingons = 0)

	uss_e_is_alive: BOOLEAN
			-- check if USS_E is alive

	uss_e_state: STRING_8
			-- return the condition of the USS_E
			-- USS_E condition state: GREEN, RED, DEAD

	uss_e_out: STRING_8
			-- display USS_E status (scores) in String format

	out: STRING_8
			-- display each unit in String, each line has 8 units

feature -- Utilities

	scan_left (host: INTEGER_32; target: INTEGER_32): BOOLEAN
			-- index `host' checks if the index `target' is on its left
			-- and in the same row returns true if the target is found.
		require
			host_positive: host > 0
			target_positive: target > 0

	scan_right (host: INTEGER_32; target: INTEGER_32): BOOLEAN
			-- index `host' checks if the index `target' is on its right
			-- and in the same row returns true if the target is found.
		require
			host_positive: host > 0
			target_positive: target > 0

	scan_up (host: INTEGER_32; target: INTEGER_32): BOOLEAN
			-- index `host' checks if the index `target' is on its up
			-- and in the same column returns true if the target is found.
		require
			host_positive: host > 0
			target_positive: target > 0

	scan_down (host: INTEGER_32; target: INTEGER_32): BOOLEAN
			-- index `host' checks if the index `target' is on its down
			-- and in the same column returns true if the target is found.
		require
			host_positive: host > 0
			target_positive: target > 0
```

```
	get_x_axis (index: INTEGER_32; mod: INTEGER_32): INTEGER_32
			-- modular calculation of x_axis(column) based on defined
			-- `mod'(number of units per line) and index in the list
		require
			index_positive: index > 0
			mod_positive: mod > 0
		ensure
			x_correct: Result = (index - 1) \\ mod

	get_y_axis (index: INTEGER_32; mod: INTEGER_32): INTEGER_32
			-- modular calculation of y_axis (row) based on defined
			-- `mod'(number of units per line) and index in the list
		require
			index_positive: index > 0
			mod_positive: mod > 0
		ensure
			y_correct: Result = (index - 1) // mod

invariant
	space_units_non_void: space_units /= Void
	count_in_range: count >= 0 and count <= Game_units
	 uss_e_id_range: uss_e_id > 0 and uss_e_id <= Game_units
	count_kl_in_range: count_klingons >= 0 and count_klingons <= Max_klingons
	count_equal_units: count = space_units.count

end -- class SPACE_MAIN_UNITS
```

Class SPACE_MAIN_UNITS_ACCESS

```
note
	description: "Summary description for SPACE_MAIN_UNITS_ACCESS. Singleton
	 accessor to SPACE_MAIN_UNITS instance."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

expanded class interface
	SPACE_MAIN_UNITS_ACCESS

create
	default_create

feature

	M: SPACE_MAIN_UNITS
			-- create singleton access point to the SPACE_MAIN_UNITS instance

invariant
	is_singleton: M = M

end -- class SPACE_MAIN_UNITS_ACCESS
```

Class SPACE_MAIN_VENGE_MODE

```
note
	description: "[
		Summary description for SPACE_MAIN_VENGE_MODE.
		A decorator class to SPACE_MAIN_UNITS enables
		VENGE_MODE: once Klingons are under the attack,
		each of them scans its enemy vertically and
		horizontally straight and attacks back the enemy.
	]"
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class interface
	SPACE_MAIN_VENGE_MODE

create
	make

feature -- attributes

	hit_count: INTEGER_32
			-- successful hit

	damage_report: STRING_8
			-- report damge to uss_e

feature -- command

	venge
			-- klingons attack starts

	reset
			-- clear history of venge mode
		ensure
			hit_reset: hit_count = 0
			report_reset: damage_report ~ create {STRING_8}.make_empty

invariant
	non_void_m: main_units /= Void

end -- class SPACE_MAIN_VENGE_MODE
```

Class SPACE_MAIN_VENGE_MODE_ACCESS

```
note
	description: "Summary description for SPACE_MAIN_VENGE_MODE_ACCESS.
		 Singleton accessor to SPACE_MAIN_UNITS instance."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

expanded class interface
	SPACE_MAIN_VENGE_MODE_ACCESS
```

```
create
	default_create

feature

	Singleton (m: SPACE_MAIN_UNITS): SPACE_MAIN_VENGE_MODE
			-- create singleton access to the SPACE_MAIN_VENGE_MODE instance
		require
			non_void_m: m /= Void
		ensure
			is_singleton: Result = Result

end -- class SPACE_MAIN_VENGE_MODE_ACCESS
```

Class SPACE_UNIT

```
note
	description: "Summary description for SPACE_UNIT. Deferred Class defines a
	 basic abstract unit for the game."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

deferred class interface
	SPACE_UNIT

feature -- SPACE_UNIT information

	name: STRING_8
			-- unit category or name
		ensure
				name /= Void

	id: INTEGER_32
			-- unit position in list
		ensure
				id > 0 and id <= Max_id

	Max_id: INTEGER_32 = 64
			-- global constant, max number a id can be

feature -- command

	set_id (index: INTEGER_32)
			-- set a new id `index'
		require
			index_positive: index > 0 and index <= Max_id
		ensure
			id_assigned: id = index
```

```
invariant
	name_non_void: name /= Void
	id_in_range: id > 0 and id <= Max_id

end -- class SPACE_UNIT
```

Class SPACE_TRACK

```
note
	description: "[
		Summary description for SPACE_TRACK. Inherit from SPACE_UNIT.
		Represent a clear galaxy space allow aircrafts to fly through.
	]"
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class interface
	SPACE_TRACK

create
	make

feature -- SPACE_TRACK attribtes

	name: STRING_8
			-- unit category or name

	id: INTEGER_32
			-- index of the collection where SPACE_TRACK resides

feature -- command

	set_id (index: INTEGER_32)
			-- set a new id `index'

invariant
	name_correct: create {STRING_8}.make_from_string (" . ") ~ name
	id_in_range: id >= 0 and id <= Max_id

end -- class SPACE_TRACK
```

Class KLINGON

```
note
	description: "[
		Summary description for KLINGON. Inherit from SPACE_TRACK.
		KLINGON, an invading Klingon warship
	]"
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"
```

```
class interface
	KLINGON

create
	make

feature -- KLINGON attriburtes

	name: STRING_8
			-- unit category or name

	id: INTEGER_32
			-- index of the collection where klingon resides

feature -- Command

	set_id (index: INTEGER_32)
			-- set a new id `index'

invariant
	name_correct: create {STRING_8}.make_from_string (" * ") ~ name
	id_in_range: id >= 0 and id <= Max_id

end -- class KLINGON
```

Class USS_ENTERPISE

```
note
	description: "[
		Summary description for USS_ENTERPRISE. Inherit from SPACE_UNIT.
		USS Enterprise controlled by the player on a mission
		to hunt down and destroy invading Klingon warships.
	]"
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class interface
	USS_ENTERPRISE

create
	make

feature -- USS_E Attributes

	name: STRING_8
			-- unit category or name

	id: INTEGER_32
			-- index of the collection where USS_E resides

	torpedoes: INTEGER_32

	energy: INTEGER_32
```

```
	shields: INTEGER_32

	condition: SPACE_UNIT_STATE
			-- 3 levels: GREEN, RED, DEAD

	alive: BOOLEAN

feature -- commands

	set_id (index: INTEGER_32)
			-- set a new id `index'

	condition_check
			-- check USS_E ship condition after every activity.
			-- e.g. move, hit, attack and get_point
		require
			condution_non_void: condition /= Void

	move
			-- USS_E moves to a new position, consume 50 energy
		ensure
			energy_correct: energy = old energy - 50

	crash
			-- USS_E crashes into a unit, lose 500 shields
		ensure
			enough_shields:
				shields = old shields - 500 implies old shields >= 500
			less_shields: shields = 0 implies old shields <= 500

	attack
			-- USS_E launches a torpedo
		require
			torpedo_not_out: torpedoes > 0
		ensure
				torpedoes = old torpedoes - 1

	hit
			-- USS_E is hit by klingon war crafts,
			-- consume 100 shields
		require
			shields_valid: shields >= 0
		ensure
			enough_h_shields:
				shields = old shields - 100 implies old shields >= 100
			less_h_shields: shields = 0 implies old shields <= 100

	get_point
			-- USS_E destroy a Klingon war craft,
			-- gain 1 torpedo, 100 energy, 100 shields
		ensure
				torpedoes = old torpedoes + 1
				energy = old energy + 100
				shields = old shields + 100
```

```
feature -- query

	is_alive: BOOLEAN
			-- check if USS_E is alive
		ensure
			alive_check: Result = (condition.state /~ "DEAD")

	condition_status: STRING
			-- report the state of USS_E
		ensure
			state_check: Result = condition.state

invariant
	name_correct: create {STRING_8}.make_from_string ("-E-") ~ name
	id_in_range: id > 0 and id <= Max_id
	condition_non_void: condition /= Void
	energy_positive: energy >= 0
	torpedoes_positive: torpedoes >= 0
	shields_positive: shields >= 0

end -- class USS_ENTERPRISE
```

Class SPACE_UNIT_STATE

```
note
	description: "Summary description for SPACE_UNIT_STATE. Deferred class defines
	              a basic abstract state class for SPACE_UNIT subclasses.
	              It maintains the state of SPACE_UNIT if applicable."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

deferred class
	SPACE_UNIT_STATE

feature -- attribute

	state: STRING_8
			-- state of the SPACE_UNIT
		deferred
		end

	update (s: SPACE_UNIT)
		require
			non_void_s: s /= Void
		deferred
		end
invariant
	state_non_void: state /= Void

end -- class SPACE_UNIT_STATE
```

Class USS_E_STATE

```
note
	description: "Summary description for USS_E_STATE. Inherited from
	              SPACE_UNIT_STATE, maintains the state of USS_ENTERPRISE."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class interface
	USS_E_STATE

create
	make

feature -- attribute

	state: STRING_8
		-- 3 levels: GREEN, RED, DEAD

feature -- cmd

	update (e: USS_ENTERPRISE)
		require else
			is_uss_e: e.name ~ "-E-"
		ensure then
			correct_state: state = e.condition.state

invariant
	condition_correct:
		    create {STRING_8}.make_from_string ("GREEN") ~ state
		 or create {STRING_8}.make_from_string ("RED") ~ state
		 or create {STRING_8}.make_from_string ("DEAD") ~ state

end -- class USS_E_STATE
```

Class USS_E_STATE_ACCESS

```
note
	description: "Summary description for USS_E_STATE_ACCESS.
	              Singleton accessor to USS_E_STATE instance"
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

expanded class interface
	USS_E_STATE_ACCESS

create
	default_create

feature

	Uss_e_state: USS_E_STATE
		-- create singleton access point to the USS_E_STATE instance
```

```
invariant
	is_singleton: Uss_e_state = Uss_e_state

end -- class USS_E_STATE_ACCESS
```

5. Contracts Implementation

Class CONTRL

```
note
	description: "Summary description for CONTRL. Controller of the game"
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class
	CONTRL

inherit
	SINGLETON

create
	make

feature {NONE} -- Initialization

	make (m: SPACE_MAIN_UNITS; v: DISPLAY)
			-- Initialization for `Current'.
		require
			non_void_m: m /= Void
			non_void_v: v /= Void
		do
			model := m
			view := v
		ensure
			model_correct: model = m
			view_correct: view = v
		end

feature {NONE} -- singleton

	The_singleton: SINGLETON
			-- the unique instance of this class
		once
			Result := Current
		end

feature {NONE} -- Attributes

	model: SPACE_MAIN_UNITS
```

```
	view: DISPLAY

	game_units: INTEGER_32
		do
			Result := model.Game_units
		end

feature -- public attribute

	r: RAND_SEQ
			-- provide random features from RAND_SEQ
		do
			create Result.make
		end

feature {NONE} -- constants

	Invalid_command: STRING_8 = " Please enter a valid input.%N"
			-- to prompt user for any invalid input

feature -- command

	start
			-- game start
		local
			input: STRING_8
			output: STRING_8
			buffer: STRING_8
			luck: INTEGER_32
			venge_mode: SPACE_MAIN_VENGE_MODE
			venge_access: SPACE_MAIN_VENGE_MODE_ACCESS
		do
			create output.make_empty
			create buffer.make_empty
			venge_mode := venge_access.Singleton (model)
			view.send (model.Report.Greeting)
			view.send (model.Report.Luck_intro)
			view.send (model.Report.Attack_warning)
			from
				create input.make_empty
				luck := (r.get_rnd.item \\ game_units) + 1
			until
				input ~ "q"
			loop
				view.send (model.out)
				view.send (output)
				view.send (model.Report.Menu)
				view.send (model.Report.Command)
				input := view.receive
				if input ~ "l" or input ~ "r"
				   or input ~ "u" or input ~ "d"
				then
					output := model.move (input)
					if model.uss_e_id = luck then
						model.uss_e_ammunition_call
```

```
                output := output + model.Report.Luck
                luck := (r.get_rnd.item \\ game_units) + 1
        end
elseif input ~ "al" or input ~ "ar"
        or input ~ "au" or input ~ "ad"
then
        output := model.attack (input)
        venge_mode.venge
        output := output + venge_mode.damage_report
        venge_mode.reset
elseif input ~ "q" then
        view.send (model.Report.Quit)
else
        buffer := Invalid_command
        output := Invalid_command
end
if input /~ "q" and buffer /~ Invalid_command
    and model.uss_e_state ~ "RED"
then
        view.send (model.out)
        view.send (output)
        output := ""
        view.send (model.Report.Red_help)
        view.send (model.Report.Command)
        input := view.receive
        if input ~ "h" then
                model.uss_e_ammunition_call
                output := model.Report.Red_call
                buffer := Invalid_command
        elseif input ~ "q" then
                view.send (model.Report.Quit)
        else
                buffer := Invalid_command
        end
end
if model.is_victory then
        view.send (model.out)
        view.send (model.Report.Victory)
        input := view.receive
        input := ""
        output := ""
        buffer := ""
        model.continue
        luck := (r.get_rnd.item \\ game_units) + 1
        view.send (model.Report.Greeting)
end
if model.uss_e_is_alive = False then
        view.send (output)
        view.send (model.Report.Lose)
        view.send (model.Report.Restart)
        view.send (model.Report.Command)
        from
                input := view.receive
        until
                input ~ "s" or input ~ "q"
```

```
                                loop
                                        view.send (Invalid_command)
                                        view.send (model.Report.Command)
                                        input := view.receive
                                end
                                if input ~ "s" then
                                        model.reset
                                        view.send (model.Report.Greeting)
                                        view.send (model.Report.Luck_intro)
                                        view.send (model.Report.Attack_warning)
                                        output := ""
                                        buffer := ""
                                        luck := (r.get_rnd.item \\ game_units) + 1
                                elseif input ~ "q" then
                                        view.send (model.Report.Quit)
                                end
                        end
                end
        end

invariant
        non_void_m: model /= Void
        non_void_v: view /= Void
        game_units_correct: model.Game_units = game_units

end -- class CONTRL
```

Class CONTRL_ACCESS

```
note
        description: "Summary description for CONTRL_ACCESS.
                      Singleton accessor to CONTRL instance"
        author: "Xing Zhao"
        date: "$Date$"
        revision: "$Revision$"

expanded class
        CONTRL_ACCESS

create
        default_create

feature

        Singleton (m: SPACE_MAIN_UNITS; v: DISPLAY): CONTRL
                        -- create singleton access to the CONTRL instance
                require
                        non_void_m: m /= Void
                        non_void_v: v /= Void
                once
                        create Result.make (m, v)
                ensure
                        is_singleton: Result = Result
                end
```

```
end -- class CONTRL_ACCESS
```

Class DISPLAY

```
note
	description: "Summary description for DISPLAY. View of the game,
	              display game view in text."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class
	DISPLAY

create
	make

feature {NONE} -- Initialization

	make
			-- Initialization for `Current'.
		do
		end

feature -- commands

	send (content: STRING_8)
			-- send the content to the view or presentation
			-- print out the game in this case
		require
			non_void_content: content /= Void
		do
			print (content)
		end

	receive: STRING_8
			-- return the received content from user/input
		do
			create Result.make_empty
			Io.readline
			Result := Io.last_string.twin
			Io.new_line
		ensure
			correct_input: Result ~ Io.last_string
		end

end -- class DISPLAY
```

Class RAND_SEQ

```
note
	description: "Summary description for RAND_SEQ. Customized RANDOM,
	              use TIME to set random seed."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class
	RAND_SEQ

create
	make

feature {NONE} -- Initialization

	make
			-- Initialization for `Current'.
			-- This computes milliseconds since midnight.
		do
			create r_time.make_now
			r_seed := r_time.hour
			r_seed := r_seed * 60 + r_time.minute
			r_seed := r_seed * 60 + r_time.second
			r_seed := r_seed * 1000 + r_time.milli_second
			create r.make
			r.set_seed (r_seed)
		end

feature {NONE} -- attributes

	r_time: TIME

	r_seed: INTEGER_32

	r: RANDOM

feature -- query

	get_seed: INTEGER_32
			-- return a random seed
		do
			Result := r_seed
		ensure
			correct_seed: Result = r_seed
		end

	get_rnd: RANDOM
			-- return a RANDOM r
		do
			Result := r
		ensure
			correct_r: Result = r
		end
```

```
	get_random_set (num: INTEGER_32; index: INTEGER_32): LIST [INTEGER_32]
			-- generate an unique set of random number in range from 0 to
			-- `index'. The size of the set is `num'.
		require
			num_positive: num > 0
			mod_positive: index > 0
		local
			r_set: LIST [INTEGER_32]
			n: INTEGER_32
		do
			create {ARRAYED_LIST [INTEGER_32]} r_set.make (num)
			from
				r.start
			until
				r.After or r_set.count = num
			loop
				n := r.item \\ index
				if not r_set.has (n) and n /= 0 then
					r_set.extend (n)
				end
				r.forth
			end
			Result := r_set
		ensure
			correct_size: Result.count = num
		end

invariant
	non_void_time: r_time /= Void
	non_void_rand: r /= Void

end -- class RAND_SEQ
```

Class MESSAGE

```
note
	description: "Summary description for MESSAGE. Game messages to notify the
	              player."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class
	MESSAGE

create
	default_create

feature -- Global Constants of game messages

	Greeting: STRING_8 = "%N Welcome to USS Enterprise, Captain!%N Your fleet is
	                      ready to commence.%N"
			-- message at the start of the game
```

```
Luck_intro: STRING_8 = " Upon you move, you may pick up some%N  supply left by
                        our Allies.%N"
            -- luck rule introduction at the start of the game

Attack_warning: STRING_8 = " Your attack will expose your position%N  and
                           attract attacks from klingons on%N  your row and
                           column.%N%N"
            -- attack warning for venge mode

Command: STRING_8 = " command: "
            -- command

Restart: STRING_8 = " To restart the game, enter 's'. Or enter 'q' to quit.%N"
            -- restart the game

Menu: STRING_8 = " [l-left  r-right  u-up  d-down  al-attack left %N ar-attack
                  right au-attack up  ad-attack down  q-quit]%N"
            -- menu of the command options

Edge: STRING_8 = " You have reached to the edge, choice another direction.%N"
            -- reach the edge of quadrant

Point: STRING_8 = " torpedo+1, energy+100, shields+100.%N"
            -- point message

Gain_point: STRING_8 = " Destoried one klingon fleet.%N torpedo+1, energy+100,
                        shields+100.%N"
            -- destroyed one Klingon

Miss: STRING_8 = " No klingons on your attacked direction.%N"
            -- no target is hit

Victory: STRING_8 = " VICTORY!%N Congratulations, Captain! You destoried all
                     invaders.%N Enter any key to start a new mission.%N"
            -- victory, clear all Klingons in the quadrant

Lose: STRING_8 = " USS Enterprise condition is DEAD.%N Mission failed!%N"
            -- lose the game, USS Enterprise condition is dead

Quit: STRING_8 = " Mission Aborted!%N"
            -- quite the game

Uss_e_err: STRING_8 = "ERR: USS_ENTERPRICE not loaded yet"
            -- USS_E load error

Uss_e_move_err: STRING_8 = "ERR: USS_ENTERPRICE not loaded yet"
            -- USS_E move error

Move: STRING_8 = " Message: move costs 50 energy%N"
            -- move cost message

Crash: STRING_8 = " Warning: crashed into a Klingon fleet, -500 shields!%N"
            -- crash into a klingon, lost 500 shields
```

```
	Red_help: STRING_8 = " Warning!%N USS Enterprise is in RED condition.%N Enter
	                      'h' to call for ammunition%N Or enter any key to pass.%N"
	                -- offer hidden option when USS_E state is RED

	Red_call: STRING_8 = " Ammunition is arrived!%N torpedo+1, energy+100,
	                       shields+100.%N"
	                -- ammunition is arrived after RED state help request

	Luck: STRING_8 = " Congratulations!%N You just picked up some supply left by
	                    our Allies.%N torpedo+1, energy+100, shields+100.%N"
	                -- hit luck by moving into a lucky space position

end -- class MESSAGE
```

Class SINGLETON

```
note
	description: "Summary description for {SINGLETON}. Deferred class defines
	              Singleton allowing multiple singleton class in a system."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

deferred class
	SINGLETON

feature {NONE}

	the_singleton: SINGLETON
	                -- the unique instance of this class
	                -- should be redefined as a once function
	                -- return Current in concrete subclass
	        deferred
	        end

invariant
	only_one_instance: Current = the_singleton

end -- class SINGLETON
```

Class SPACE_MAIN_UNITS

```
note
	description: "Summary description for SPACE_MAIN_UNITS. Model of the game.
	              The Main structure that contains all game UNITs and operation
	               features."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"
```

```
class
	SPACE_MAIN_UNITS

inherit
	ANY
		redefine
			out
		end

	SINGLETON
		redefine
			out
		end

create {SPACE_MAIN_UNITS_ACCESS}
	make

feature {NONE} -- Initialization

	make
			-- Initialization for `Current'.
		do
			count := 0
			count_klingons := 0
			create {ARRAYED_LIST [SPACE_UNIT]} space_units.make (Game_units)
			space_units.compare_objects
			load_space_track
			load_klingons
			load_uss_e
		end

feature {NONE} -- singleton

	The_singleton: SINGLETON
			-- the unique instance of this class
		once
			Result := Current
		end

feature -- model attributes

	count: INTEGER_32
			-- total number of SPACE_UNITs

	count_klingons: INTEGER_32
			-- number of klingons warships

	space_units: LIST [SPACE_UNIT]
			-- list contains all SPACE_UNITS

	uss_e_id: INTEGER_32
			-- index of USS_E in space_units list

feature -- global constants
```

```
	Mode: INTEGER_32 = 8
			-- mode number, game has 8 X 8 quadrant

	Game_units: INTEGER_32 = 64
			-- the total game units

	Max_klingons: INTEGER_32 = 10
			-- max number of klingons

feature -- model operations

	reset
			-- Reset model state.
		do
			make
			uss_e_update
		end

	continue
			-- Start a new level game after Victory
			-- USS_E carries the same information from previous game
		local
			uss_e_t: USS_ENTERPRISE
		do
			if attached {USS_ENTERPRISE} space_units [uss_e_id] as uss_t then
				uss_e_t := uss_t
				reset
				space_units [uss_e_id] := uss_e_t
			else
				print ("continue error!")
			end
		end

feature -- Report messages

	Report: MESSAGE
			-- report message, create singleton access to the MESSAGE
			 -- instance
		once
			create Result
		end

feature -- Game units loading commands

	load_space_track
			-- fill the list with SPACE_TRACKs
		require
			non_void_list: space_units /= Void
		local
			st: SPACE_UNIT
			i: INTEGER_32
		do
			from
				i := 1
				space_units.start
```

```
            until
                  space_units.count = Game_units
            loop
                  create {SPACE_TRACK} st.make (i)
                  space_units.extend (st)
                  count := count + 1
                  i := i + 1
                  space_units.forth
            end
      ensure
            list_full_loaded: count = Game_units
      end

load_klingons
            -- fill the KLINGONs with random index to the list
      local
            rand: LIST [INTEGER_32]
            r: RAND_SEQ
            k: SPACE_UNIT
            i: INTEGER_32
      do
            create r.make
            rand := r.get_random_set (Max_klingons, Game_units)
            from
                  rand.start
            until
                  rand.after
            loop
                  i := rand.item
                  create {KLINGON} k.make (i)
                  space_units [i] := k
                  count_klingons := count_klingons + 1
                  rand.forth
            end
      ensure
            klingons_load_correct: count_klingons = Max_klingons
      end

load_uss_e
            -- insert USS_E randomly into a SPACE_TRACK
      local
            uss_e: SPACE_UNIT
            i: INTEGER_32
            r: RANDOM
            r_num: RAND_SEQ
      do
            create r_num.make
            r := r_num.get_rnd
            from
                  r.start
            until
                  r.After or i = -1
            loop
                  i := (r.item \\ Game_units) + 1
                  if space_units [i].name ~ " . " then
```

```
                                create {USS_ENTERPRISE} uss_e.make (i)
                                space_units [i] := uss_e
                                uss_e_id := i
                                i := -1
                        end
                        r.forth
                end
        ensure
                uss_e_id_in_range: uss_e_id > 0 and uss_e_id <= Game_units
        end

feature -- commands

        klingons_decrease
                -- USS_E destorys one klingons
        require
                positive_count_kling: count_klingons > 0
        do
                count_klingons := count_klingons - 1
        ensure
                        count_klingons = old count_klingons - 1
        end

        uss_e_update
                -- Perform update to the USS_E condition.
        do
                if attached {USS_ENTERPRISE} space_units [uss_e_id] as uss_t then
                        uss_t.condition_check
                end
        end

        uss_e_ammunition_call
                -- USS_E calls for more ammunition under RED state or luck point
        do
                if attached {USS_ENTERPRISE} space_units [uss_e_id] as uss_t then
                        uss_t.get_point
                end
        end

        hit_target (invader: INTEGER_32)
                -- successfully destories a klingon at index `invader'
        require
                invader_in_range: invader > 0 and invader <= count
        do
                if attached {USS_ENTERPRISE} space_units [uss_e_id] as uss_t then
                        uss_t.get_point
                        uss_t.attack
                        uss_t.condition_check
                        space_units [invader] :=
                                        create {SPACE_TRACK}.make (invader)
                        klingons_decrease
                end
        end

        miss_target
```

```
                -- missed taget, USS_E wastes 1 torpedo
        do
                if attached {USS_ENTERPRISE} space_units [uss_e_id] as uss_t then
                        uss_t.attack
                end
        end

attack (direction: STRING_8): STRING_8
                -- USS_E attacks klingon in the targated direction
        require
                a_direction_non_void: direction /= Void
                a_direction_correct: direction ~ "al" or direction ~ "ar"
                                or direction ~ "au" or direction ~ "ad"
        local
                sc_index: INTEGER_32
        do
                sc_index := uss_e_id
                create {STRING_8} Result.make_empty
                if direction ~ "al" then
                        from
                                sc_index := sc_index - 1
                        until
                                sc_index < 1 or Result ~ Report.Gain_point
                        loop
                                if space_units [sc_index].name ~ " * " and
                                   scan_left (uss_e_id, sc_index)
                                then
                                        hit_target (sc_index)
                                        Result := Report.Gain_point
                                end
                                sc_index := sc_index - 1
                        end
                elseif direction ~ "ar" then
                        from
                                sc_index := sc_index + 1
                        until
                                sc_index > count or Result ~ Report.Gain_point
                        loop
                                if space_units [sc_index].name ~ " * " and
                                   scan_right (uss_e_id, sc_index)
                                then
                                        hit_target (sc_index)
                                        Result := Report.Gain_point
                                end
                                sc_index := sc_index + 1
                        end
                elseif direction ~ "au" then
                        from
                                sc_index := sc_index - 1
                        until
                                sc_index < 1 or Result ~ Report.Gain_point
                        loop
                                if space_units [sc_index].name ~ " * " and
                                   scan_up (uss_e_id, sc_index)
```

```
                        then
                              hit_target (sc_index)
                              Result := Report.Gain_point
                        end
                        sc_index := sc_index - 1
                  end
            elseif direction ~ "ad" then
                  from
                        sc_index := sc_index + 1
                  until
                        sc_index > count or Result ~ Report.Gain_point
                  loop
                        if space_units [sc_index].name ~ " * " and
                           scan_down (uss_e_id, sc_index)
                        then
                              hit_target (sc_index)
                              Result := Report.Gain_point
                        end
                        sc_index := sc_index + 1
                  end
            end
            if Result.is_empty then
                  miss_target
                  Result := Report.Miss
            end
      end

move (direction: STRING_8): STRING_8
            -- move USS_E to the `direction' one unit position
      require
            m_direction_non_void: direction /= Void
            m_direction_correct: direction ~ "l" or direction ~ "r"
                              or direction ~ "u" or direction ~ "d"
      do
            create Result.make_empty
            if direction ~ "l" and uss_e_id > 1 then
                  if space_units [uss_e_id - 1].name ~ " * " then
                        uss_e_crash (uss_e_id - 1)
                        Result := Report.Crash
                  else
                        uss_e_swape (uss_e_id - 1)
                  end
            elseif direction ~ "r" and uss_e_id < count then
                  if space_units [uss_e_id + 1].name ~ " * " then
                        uss_e_crash (uss_e_id + 1)
                        Result := Report.Crash
                  else
                        uss_e_swape (uss_e_id + 1)
                  end
            elseif direction ~ "u" and (uss_e_id - 1) // Mode > 0 then
                  if space_units [uss_e_id - Mode].name ~ " * " then
                        uss_e_crash (uss_e_id - Mode)
                        Result := Report.Crash
                  else
                        uss_e_swape (uss_e_id - Mode)
```

```
                        end
                    elseif direction ~ "d" and (uss_e_id - 1) // Mode < Mode - 1 then
                        if space_units [uss_e_id + Mode].name ~ " * " then
                            uss_e_crash (uss_e_id + Mode)
                            Result := Report.Crash
                        else
                            uss_e_swape (uss_e_id + Mode)
                        end
                    else
                        Result := Report.Edge
                    end
                    if Result.is_empty then
                        if attached {USS_ENTERPRISE} space_units [uss_e_id] as
                            uss_t
                        then
                            uss_t.move
                            Result := Report.Move
                        else
                            Result := Report.Uss_e_err
                        end
                    end
                end

        uss_e_swape (index: INTEGER_32)
                    -- swape USS_E unit with the unit at `index'
                require
                    swap_index_in_range: index > 0 and index <= count
                do
                    space_units.go_i_th (uss_e_id)
                    space_units.swap (index);
                    space_units [index].set_id (index);
                    space_units [uss_e_id].set_id (uss_e_id)
                    uss_e_id := index
                ensure
                    index_assigned: uss_e_id = index
                    uss_e_id_assigned: space_units [uss_e_id].id = index
                    uss_e_valid: space_units [uss_e_id].name ~ "-E-"
                end

        uss_e_crash (index: INTEGER_32)
                    -- USS_E crashes into the unit at `index' and
                    -- replaces the unit at `index'
                require
                    clash_index_in_range: index > 0 and index <= count
                do
                    space_units [index] := create {SPACE_TRACK}.make (index)
                    uss_e_swape (index)
                    klingons_decrease
                    if attached {USS_ENTERPRISE} space_units [uss_e_id] as uss_t then
                        uss_t.crash
                    end
                end

feature -- queries
```

```
quadrant_out (n: INTEGER_32): STRING_8
		-- quadrant information in String format
		-- `n', size of the quadrant in n x n
	require
		n_positive: n > 0
	local
		col: INTEGER_32
	do
		create Result.make_from_string (" quadrant%N")
		from
			col := 1
		until
			col > space_units.count
		loop
			Result := Result + space_units [col].name
			if ((col - 1) \\ n) = n - 1 then
				Result := Result + "%N"
			end
			col := col + 1
		end
		Result := Result + "%N"
	end

is_victory: BOOLEAN
		-- check if all kinglons are destoryed
	do
		if count_klingons = 0 then
			Result := True
		end
	ensure
		victory_check: Result = (count_klingons = 0)
	end

uss_e_is_alive: BOOLEAN
		-- check if USS_E is alive
	do
		if attached {USS_ENTERPRISE} space_units [uss_e_id] as uss_t then
			Result := uss_t.is_alive
		end
	end

uss_e_state: STRING_8
		-- return the condition of the USS_E
		-- USS_E condition state: GREEN, RED, DEAD
	do
		create Result.make_empty
		if attached {USS_ENTERPRISE} space_units [uss_e_id] as uss_t then
			Result := uss_t.condition_status
		else
			Result := Report.Uss_e_err
		end
	end

uss_e_out: STRING_8
		-- display USS_E status (scores) in String format
```

```
            do
                  create Result.make_empty
                  if attached {USS_ENTERPRISE} space_units [uss_e_id] as uss_t then
                        Result := " condition %T" + uss_t.condition_status + "%N"
                        Result := Result + " torpedoes %T" + uss_t.torpedoes.out
                                         + "%N"
                        Result := Result + " energy   %T" + uss_t.energy.out
                                         + "%N"
                        Result := Result + " shields  %T" + uss_t.shields.out
                                         + "%N"
                        Result := Result + " klingons  %T" + count_klingons.out
                                         + "%N"
                  else
                        Result := Report.Uss_e_err
                  end
            end

      out: STRING_8
                  -- display each unit in String, each line has 8 units
            do
                  create Result.make_from_string ("")
                  Result := uss_e_out
                  Result := Result + quadrant_out (Mode)
            end

feature -- Utilities

      scan_left (host: INTEGER_32; target: INTEGER_32): BOOLEAN
                  -- index `host' checks if the index `target' is on its left
                  -- and in the same row returns true if the target is found.
            require
                  host_positive: host > 0
                  target_positive: target > 0
            do
                  if target < host and
                     get_y_axis (target, Mode) = get_y_axis (host, Mode)
                   then
                        Result := True
                  end
            end

      scan_right (host: INTEGER_32; target: INTEGER_32): BOOLEAN
                  -- index `host' checks if the index `target' is on its right
                  -- and in the same row returns true if the target is found.
            require
                  host_positive: host > 0
                  target_positive: target > 0
            do
                  if target > host and
                     get_y_axis (target, Mode) = get_y_axis (host, Mode)
                   then
                        Result := True
                  end
            end
```

```
scan_up (host: INTEGER_32; target: INTEGER_32): BOOLEAN
            -- index `host' checks if the index `target' is on its up
            -- and in the same column returns true if the target is found.
      require
            host_positive: host > 0
            target_positive: target > 0
      do
            if target < host and
               get_x_axis (target, Mode) = get_x_axis (host, Mode)
             then
                  Result := True
            end
      end

scan_down (host: INTEGER_32; target: INTEGER_32): BOOLEAN
            -- index `host' checks if the index `target' is on its down
            -- and in the same column returns true if the target is found.
      require
            host_positive: host > 0
            target_positive: target > 0
      do
            if target > host and
               get_x_axis (target, Mode) = get_x_axis (host, Mode)
                                                      then
                  Result := True
            end
      end

get_x_axis (index: INTEGER_32; mod: INTEGER_32): INTEGER_32
            -- calcuate x_axis,column based on defined `mod'(number
            -- of units per line) and index in the list
      require
            index_positive: index > 0
            mod_positive: mod > 0
      do
            Result := (index - 1) \\ mod
      ensure
            x_correct: Result = (index - 1) \\ mod
      end

get_y_axis (index: INTEGER_32; mod: INTEGER_32): INTEGER_32
            -- calcuate y_axis,row based on defined `mod'(number
            -- of units per line) and index in the list
      require
            index_positive: index > 0
            mod_positive: mod > 0
      do
            Result := (index - 1) // mod
      ensure
            y_correct: Result = (index - 1) // mod
      end
```

```
invariant
	space_units_non_void: space_units /= Void
	count_in_range: count >= 0 and count <= Game_units
	 uss_e_id_range: uss_e_id > 0 and uss_e_id <= Game_units
	count_kl_in_range: count_klingons >= 0 and count_klingons <= Max_klingons
	count_equal_units: count = space_units.count

end -- class SPACE_MAIN_UNITS
```

Class SPACE_MAIN_UNITS_ACCESS

```
note
	description: "Summary description for SPACE_MAIN_UNITS_ACCESS.
	              Singleton accessor to SPACE_MAIN_UNITS instance"
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

expanded class
	SPACE_MAIN_UNITS_ACCESS

create
	default_create

feature

	M: SPACE_MAIN_UNITS
			-- create singleton access to the SPACE_MAIN_UNITS instance
		once
			create Result.make
		end

invariant
	is_singleton: M = M

end -- class SPACE_MAIN_UNITS_ACCESS
```

Class SPACE_MAIN_VENGE_MODE

```
note
	description: "[
		Summary description for SPACE_MAIN_VENGE_MODE.
		A decorator class to SPACE_MAIN_UNITS enables
		VENGE_MODE: once Klingons are under the attack,
		each of them scans its enemy vertically and
		horizontally straight and attacks back the enemy.
	]"
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"
```

```
class
	SPACE_MAIN_VENGE_MODE

inherit
	SINGLETON

create
	make

feature {NONE} -- Initialization

	make (m: SPACE_MAIN_UNITS)
			-- Initialization for `Current'.
		do
			main_units := m
			damage_report := create {STRING_8}.make_empty
		ensure
			correct_main_units: main_units = m
		end

feature {NONE} -- singleton

	The_singleton: SINGLETON
			-- the unique instance of this class
		once
			Result := Current
		end

feature {NONE} -- attributes

	main_units: SPACE_MAIN_UNITS

	Hundred: INTEGER_32 = 100
			-- hit by a klingon costs 100 shield

feature -- attributes

	hit_count: INTEGER_32
			-- successful hit

	damage_report: STRING_8
			-- report damge to uss_e

feature -- command

	venge
			-- klingons attack starts
		local
			uss_e: INTEGER_32
			list: LIST [SPACE_UNIT]
		do
			uss_e := main_units.uss_e_id
			list := main_units.space_units
			across
				list as k
```

```
                        loop
                              if attached {KLINGON} k.item as vange_k then
                                    if main_units.scan_left (vange_k.id, uss_e) or
                                       main_units.scan_right (vange_k.id, uss_e) or
                                       main_units.scan_up (vange_k.id, uss_e) or
                                       main_units.scan_down (vange_k.id, uss_e)
                                    then
                                          hit_count := hit_count + 1
                                    end
                              end
                        end

                        if hit_count > 0 then
                              if attached {USS_ENTERPRISE} list [main_units.uss_e_id] as
                                          uss_t
                               then
                                    across
                                          1 |..| hit_count as c
                                    loop
                                          uss_t.hit
                                    end
                                    damage_report := " Warning!%N You are attacked by "
                                                   + hit_count.out + " klingons: -"
                                                   + (hit_count * Hundred).out
                                                   + " shields%N"
                              else
                                    print (" link to USS_E error in venge mode.")
                              end
                        end
                  end

            reset
                        -- clear history of venge mode
                  do
                        hit_count := 0
                        damage_report := create {STRING_8}.make_empty
                  ensure
                        hit_reset: hit_count = 0
                        report_reset: damage_report ~ create {STRING_8}.make_empty
                  end

invariant
      non_void_m: main_units /= Void

end -- class SPACE_MAIN_VENGE_MODE
```

Class SPACE_MAIN_VENGE_MODE_ACCESS

```
note
      description: "Summary description for SPACE_MAIN_VENGE_MODE_ACCESS.
                    Singleton accessor to SPACE_MAIN_UNITS instance"
      author: "Xing Zhao"
      date: "$Date$"
      revision: "$Revision$"
```

```
expanded class
	SPACE_MAIN_VENGE_MODE_ACCESS

create
	default_create

feature

	Singleton (m: SPACE_MAIN_UNITS): SPACE_MAIN_VENGE_MODE
			-- create singleton access to the SPACE_MAIN_VENGE_MODE instance
		require
			non_void_m: m /= Void
		once
			create Result.make (m)
		ensure
			is_singleton: Result = Result
		end

end -- class SPACE_MAIN_VENGE_MODE_ACCESS
```

Class SPACE_UNIT

```
note
	description: "Summary description for SPACE_UNIT. Deferred Class defines a
	 basic abstract unit for the game."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

deferred class
	SPACE_UNIT

feature -- SPACE_UNIT information

	name: STRING_8
			-- unit category or name
		deferred
		ensure
				name /= Void
		end

	id: INTEGER_32
			-- unit position in list
		deferred
		ensure
				id > 0 and id <= Max_id
		end

	Max_id: INTEGER_32 = 64
			-- global constant, max number a id can be

feature -- command
```

```
	set_id (index: INTEGER_32)
			-- set a new id `index'
		require
			index_positive: index > 0 and index <= Max_id
		deferred
		ensure
			id_assigned: id = index
		end

invariant
	name_non_void: name /= Void
	id_in_range: id > 0 and id <= Max_id

end -- class SPACE_UNIT
```

Class SPACE_TRACK

```
note
	description: "[
		Summary description for SPACE_TRACK. Inherit from SPACE_UNIT.
		It represents a clear galaxy space that allows aircrafts fly through.
	]"
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class
	SPACE_TRACK

inherit
	SPACE_UNIT

create
	make

feature {NONE} -- Initialization

	make (a_id: INTEGER_32)
			-- create a basic space track
		require
			id_in_range: a_id > 0 and id <= Max_id
		do
			create name.make_from_string (" . ")
			id := a_id
		ensure
			name_correct: create {STRING_8}.make_from_string (" . ") ~ name
			id_correct: id = a_id
		end

feature -- SPACE_TRACK attribtes

	name: STRING_8
			-- unit category or name
```

```
	id: INTEGER_32
			-- index of the collection where SPACE_TRACK resides

feature -- command

	set_id (index: INTEGER_32)
			-- set a new id `index'
		do
			id := index
		end

invariant
	name_correct: create {STRING_8}.make_from_string (" . ") ~ name
	id_in_range: id >= 0 and id <= Max_id

end -- class SPACE_TRACK
```

Class KLINGON

```
note
	description: "[
		Summary description for KLINGON. Inherit from SPACE_TRACK.
		KLINGON, an invading Klingon warship
	]"
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class
	KLINGON

inherit
	SPACE_UNIT

create
	make

feature {NONE} -- Initialization

	make (a_id: INTEGER_32)
			-- create a klingon war craft
		require
			a_id_in_range: a_id > 0 and a_id <= Max_id
		do
			create name.make_from_string (" * ")
			id := a_id
		ensure
			name_correct: create {STRING_8}.make_from_string (" * ") ~ name
			id_correct: id = a_id
		end

feature -- KLINGON attriburtes
```

```
	name: STRING_8
			-- unit category or name

	id: INTEGER_32
			-- index of the collection where klingon resides

feature -- Command

	set_id (index: INTEGER_32)
			-- set a new id `index'
		do
			id := index
		end

invariant
	name_correct: create {STRING_8}.make_from_string (" * ") ~ name
	id_in_range: id >= 0 and id <= Max_id

end -- class KLINGON
```

Class USS_ENTERPISE

```
note
	description: "[
		Summary description for USS_ENTERPRISE. Inherit from SPACE_UNIT.
		USS Enterprise controlled by the player on a mission
		to hunt down and destroy invading Klingon warships.
	]"
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class
	USS_ENTERPRISE

inherit
	SPACE_UNIT

create
	make

feature {NONE} -- Initialization

	make (a_id: INTEGER_32)
			-- Initialization for `Current', a new USS_E takes off.
		require
			a_id_in_range: a_id > 0 and a_id <= Max_id
		local
			state_access: USS_E_STATE_ACCESS
		do
			create name.make_from_string ("-E-")
			id := a_id
			torpedoes := 10
			energy := 2000
```

```
                shields := 1000
                condition := state_access.Uss_e_state
                alive := True
        ensure
                name_correct: create {STRING_8}.make_from_string ("-E-") ~ name
                id_correct: id = a_id
                is_alive: alive = True
        end

feature -- USS_E Attributes

        name: STRING_8
                -- unit category or name

        id: INTEGER_32
                -- index of the collection where USS_E resides

        torpedoes: INTEGER_32

        energy: INTEGER_32

        shields: INTEGER_32

        condition: SPACE_UNIT_STATE
                -- 3 levels: GREEN, RED, DEAD

        alive: BOOLEAN

feature -- commands

        set_id (index: INTEGER_32)
                -- set a new id `index'
        do
                id := index
        end

        condition_check
                -- check USS_E ship condition after every activity.
                -- e.g. move, hit, attack and get_point
        require
                condution_non_void: condition /= Void
        do
                condition.update (Current)
        end

        move
                -- USS_E moves to a new position, consume 50 energy
        do
                energy := energy - 50
                condition_check
        ensure
                energy_correct: energy = old energy - 50
        end

        crash
```

```
		-- USS_E crashes into a unit, lose 500 shields
	do
		if shields < 500 then
			shields := 0
		else
			shields := shields - 500
		end
		condition_check
	ensure
		enough_shields:
				shields = old shields - 500 implies old shields >= 500
		less_shields: shields = 0 implies old shields <= 500
	end

attack
		-- USS_E launches a torpedo
	require
		torpedo_not_out: torpedoes > 0
	do
		torpedoes := torpedoes - 1
		condition_check
	ensure
			torpedoes = old torpedoes - 1
	end

hit
		-- USS_E is hit by klingon war crafts,
		-- consume 100 shields
	require
		shields_valid: shields >= 0
	do
		if shields < 100 then
			shields := 0
		else
			shields := shields - 100
		end
	condition_check
	ensure
		enough_h_shields:
				shields = old shields - 100 implies old shields >= 100
		less_h_shields: shields = 0 implies old shields <= 100
	end

get_point
		-- USS_E destory a klingon war craft,
		-- gain 1 torpedo, 100 energy, 100 shields
	do
		torpedoes := torpedoes + 1
		energy := energy + 100
		shields := shields + 100
		condition_check
	ensure
			torpedoes = old torpedoes + 1
			energy = old energy + 100
			shields = old shields + 100
```

```
            end

feature -- query

      is_alive: BOOLEAN
                  -- check if USS_E is alive
            do
                  condition_check
                  if condition.state /~ "DEAD" then
                        Result := True
                  end
            ensure
                  alive_check: Result = (condition.state /~ "DEAD")
            end

      condition_status: STRING
                  -- report the state of USS_E
            do
                  condition_check
                  Result := condition.state
            ensure
                  state_check: Result = condition.state

invariant
      name_correct: create {STRING_8}.make_from_string ("-E-") ~ name
      id_in_range: id > 0 and id <= Max_id
      condition_non_void: condition /= Void
      energy_positive: energy >= 0
      torpedoes_positive: torpedoes >= 0
      shields_positive: shields >= 0

end -- class USS_ENTERPRISE
```

Class SPACE_UNIT_STATE

```
note
      description: "Summary description for SPACE_UNIT_STATE. Deferred class defines
                    a basic abstract state class for SPACE_UNIT subclasses. It
                    maintains the state of SPACE_UNIT if applicable."
      author: "Xing Zhao"
      date: "$Date$"
      revision: "$Revision$"

deferred class
      SPACE_UNIT_STATE

feature -- attribute

      state: STRING_8
                  -- state of the SPACE_UNIT
            deferred
            end
```

```
	update (s: SPACE_UNIT)
		require
			non_void_s: s /= Void
		deferred
		end
invariant
	state_non_void: state /= Void

end -- class SPACE_UNIT_STATE
```

Class USS_E_STATE

```
note
	description: "Summary description for USS_E_STATE. Inherited from
	              SPACE_UNIT_STATE. Maintain the state of USS_ENTERPRISE."
	author: "Xing Zhao"
	date: "$Date$"
	revision: "$Revision$"

class
	USS_E_STATE

inherit
	SPACE_UNIT_STATE

	SINGLETON

create
	make

feature {NONE} -- Initialization

	make
		do
			state := "GREEN"
		end

feature {NONE} -- singleton

	The_singleton: SINGLETON
			-- the unique instance of this class
		once
			Result := Current
		end

feature -- attribute

	state: STRING_8
			-- 3 levels: GREEN, RED, DEAD

feature -- cmd

	update (e: USS_ENTERPRISE)
		require else
```

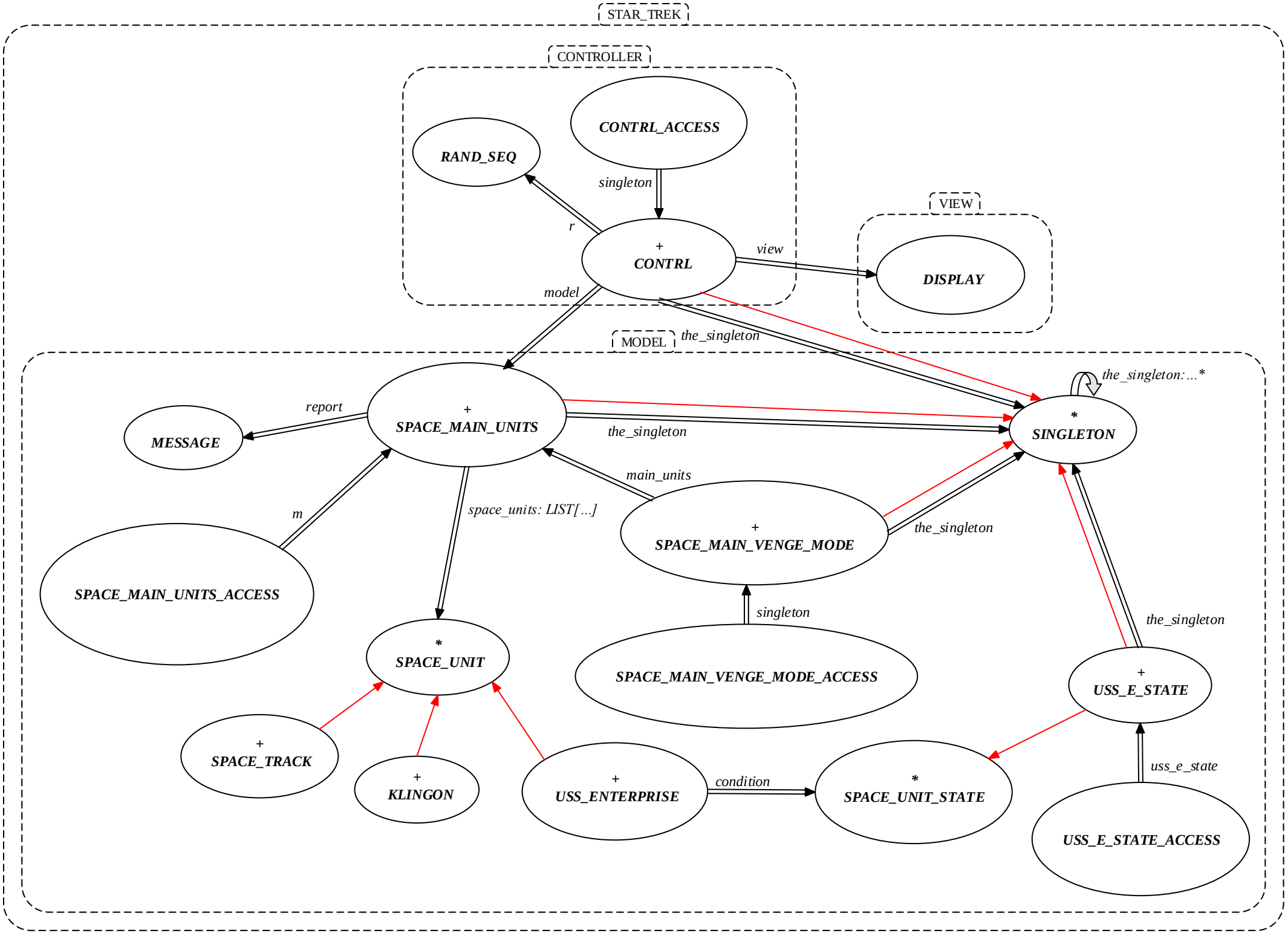
```
                is_uss_e: e.name ~ "-E-"
        do
                if e.energy <= 0 or e.torpedoes <= 0 or e.shields <= 0 then
                        state := "DEAD"
                elseif e.energy < 600 or e.torpedoes < 3 or e.shields <= 500 then
                        state := "RED"
                else
                        state := "GREEN"
                end
        ensure then
                correct_state: state = e.condition.state
        end

invariant
        condition_correct: create {STRING_8}.make_from_string ("GREEN") ~ state
                        or create {STRING_8}.make_from_string ("RED") ~ state
                        or create {STRING_8}.make_from_string ("DEAD") ~ state

end -- class USS_E_STATE
```

Class USS_E_STATE_ACCESS

```
note
        description: "Summary description for USS_E_STATE_ACCESS.
                      Singleton accessor to USS_E_STATE instance."
        author: "Xing Zhao"
        date: "$Date$"
        revision: "$Revision$"

expanded class
        USS_E_STATE_ACCESS

create
        default_create

feature

        Uss_e_state: USS_E_STATE
                        -- create singleton access to the USS_E_STATE
                once
                        create Result.make
                end

invariant
        is_singleton: Uss_e_state = Uss_e_state

end -- class USS_E_STATE_ACCESS
```

6. Design Patterns

BON Diagram Overview


STAR_TREK
CONTROLLER
CONTRL_ACCESS
RAND_SEQ
singleton
r
+
CONTRL
view
VIEW
DISPLAY
model
MODEL
the_singleton
the_singleton:...*
report
+
SPACE_MAIN_UNITS
*
SINGLETON
MESSAGE
the_singleton
main_units
m
space_units: LIST[...]
+
SPACE_MAIN_VENGE_MODE
the_singleton
SPACE_MAIN_UNITS_ACCESS
singleton
the_singleton
*
SPACE_UNIT
SPACE_MAIN_VENGE_MODE_ACCESS
+
USS_E_STATE
+
SPACE_TRACK
+
KLINGON
+
USS_ENTERPRISE
condition
*
SPACE_UNIT_STATE
uss_e_state
USS_E_STATE_ACCESS

Overview

When it comes to design software, the first thing is to consider the main structure of the application and then choose an appropriate pattern. For a game application, MVC (model-view-controller) pattern is conventionally considered since its architectural fits well to implement user interfaces. It divides a game into three parts, model, view and controller which is convenient for troubleshooting, reuse and scalability.

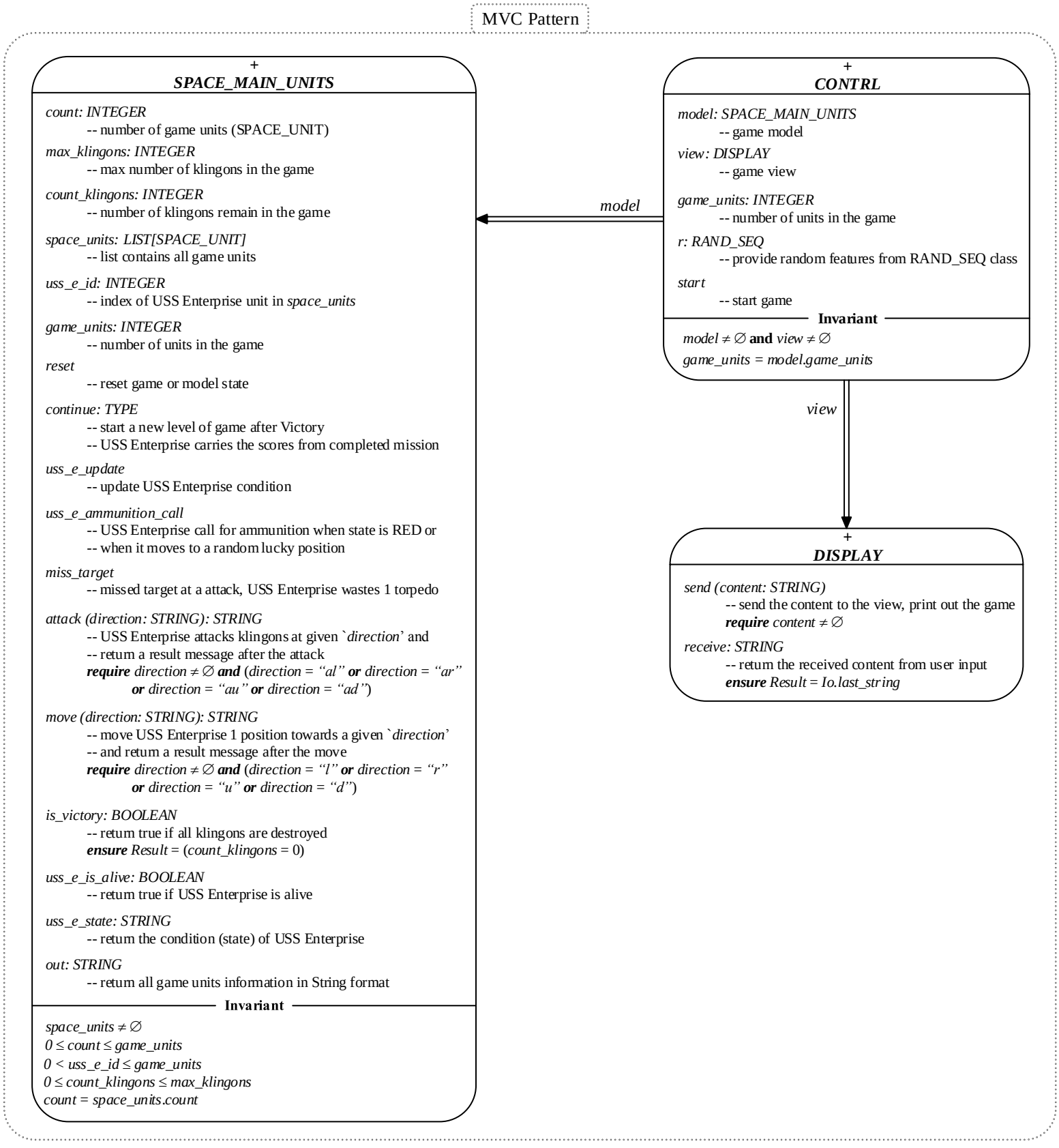
In order to keep all classes organized during the implementation and easy to maintain in the future, I created three clusters, Controller, Model and View to store the classes based on their functional purpose.

Controller cluster contains classes that serve as or to the game controller. The random features are exclusively implemented into an independent class RAND_SEQ which is aggregated into the controller class. Then, this random class can be used by other classes which require random number. Separating the random features from controller benefits the testing, future maintenance and troubleshooting. The problem can be easily isolated between controller and RAND_SEQ. Especially in the case that many classes from model cluster are using random features. In addition, it does not require controller class to be instantiated for other classes to use the random features. Hence, this isolation helps on testing the specific class or two classes in a relation from same or different modules.

View cluster includes classes display game to user and take user input. All the validations of user input are done in the controller class so that the view classes are reusable and adaptable to other games.

Model cluster keeps business or game logic and data. The game theme units and structure classes reside here. A sub cluster unit is created to organize all theme units since there are more units to be added as the game grows.

In this game, six design patterns are incorporated based on the game specifications and how well their advantages fit into the specification. Details of how the design patterns are selected and implemented are described in the following sections.


MVC Pattern
+
SPACE_MAIN_UNITS
count: INTEGER
-- number of game units (SPACE_UNIT)
max_klingons: INTEGER
-- max number of klingons in the game
count_klingons: INTEGER
-- number of klingons remain in the game
space_units: LIST[SPACE_UNIT]
-- list contains all game units
uss_e_id: INTEGER
-- index of USS Enterprise unit in space_units
game_units: INTEGER
-- number of units in the game
reset
-- reset game or model state
continue: TYPE
-- start a new level of game after Victory
-- USS Enterprise carries the scores from completed mission
uss_e_update
-- update USS Enterprise condition
uss_e_ammunition_call
-- USS Enterprise call for ammunition when state is RED or
-- when it moves to a random lucky position
miss_target
-- missed target at a attack, USS Enterprise wastes 1 torpedo
attack (direction: STRING): STRING
-- USS Enterprise attacks klingons at given `direction' and
-- return a result message after the attack
require direction ≠ ∅ and (direction = "al" or direction = "ar"
or direction = "au" or direction = "ad")
move (direction: STRING): STRING
-- move USS Enterprise 1 position towards a given `direction'
-- and return a result message after the move
require direction ≠ ∅ and (direction = "l" or direction = "r"
or direction = "u" or direction = "d")
is_victory: BOOLEAN
-- return true if all klingons are destroyed
ensure Result = (count_klingons = 0)
uss_e_is_alive: BOOLEAN
-- return true if USS Enterprise is alive
uss_e_state: STRING
-- return the condition (state) of USS Enterprise
out: STRING
-- return all game units information in String format
Invariant
space_units ≠ ∅
0 ≤ count ≤ game_units
0 < uss_e_id ≤ game_units
0 ≤ count_klingons ≤ max_klingons
count = space_units.count
model
+
CONTRL
model: SPACE_MAIN_UNITS
-- game model
view: DISPLAY
-- game view
game_units: INTEGER
-- number of units in the game
r: RAND_SEQ
-- provide random features from RAND_SEQ class
start
-- start game
Invariant
model ≠ ∅ and view ≠ ∅
game_units = model.game_units
view
+
DISPLAY
send (content: STRING)
-- send the content to the view, print out the game
require content ≠ ∅
receive: STRING
-- return the received content from user input
ensure Result = Io.last_string

MVC pattern

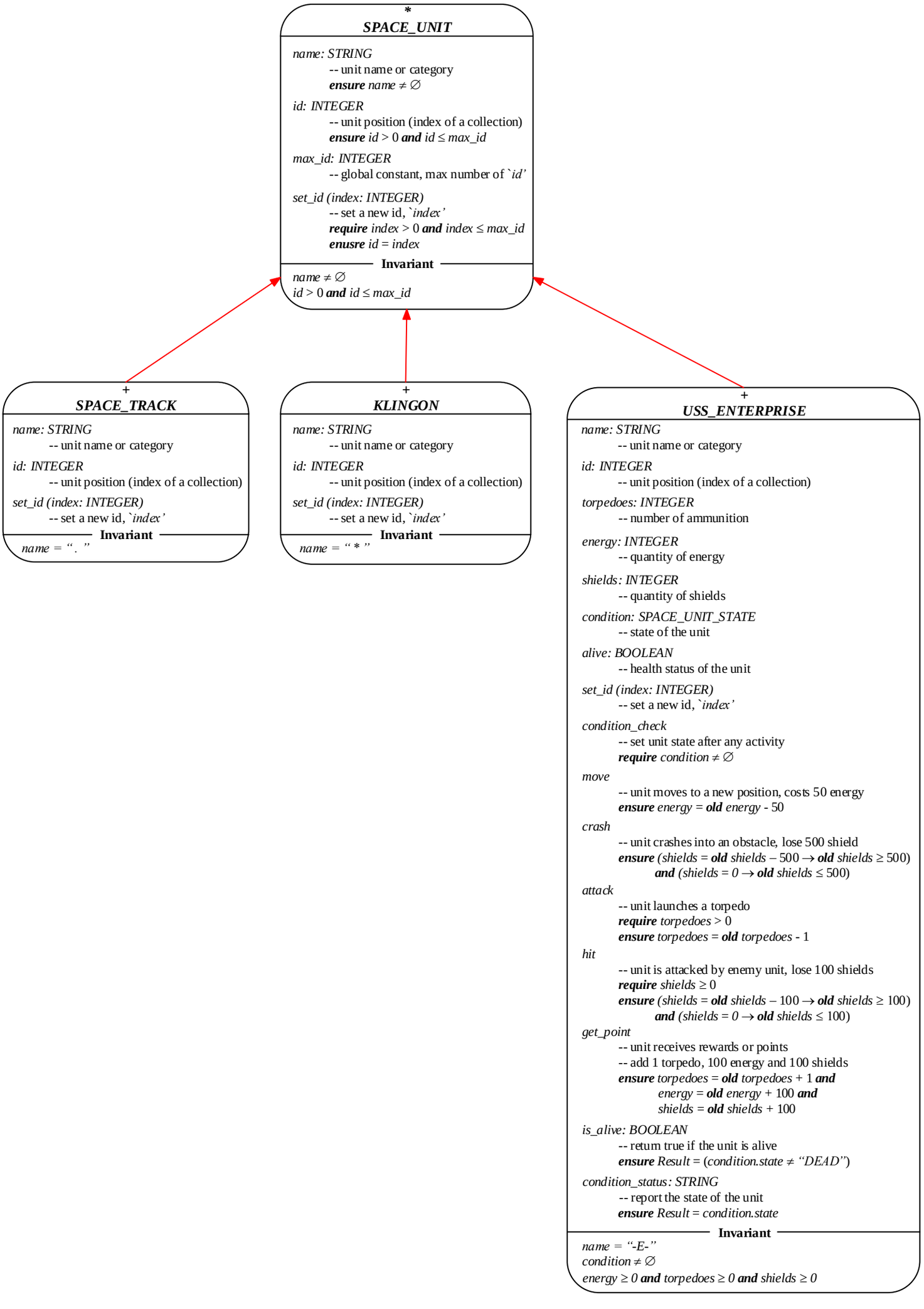
MVC is chosen simply due to the game is a user interactive application and MVC is very popular on implementing the user interface application. However, MCM was the pattern I considered first as I could print out the game directly from the model instead of using the view. However, when it comes to the implementation of user input, I realize that the model would have to deal with the validation on the input which leads to more complication in the class and may form a spaghetti code. Hence, MVC is finally selected.

Model is only responsible for game logic and data unit. Controller deals with the model and view respectively. But model does not interact with view directly. View interacts with user; it displays view to the user and takes user input. Controller sends and receives the content to and from view as well as validates the user input before it passes the validated input data to the model. When model receives the data from controller, it processes the data and returns the result to controller if applicable. Controller then passes the result to view if applicable.

Controller is implemented in singleton as per specification. The game only requires one controller instance in the system.

The model class, SPACE_MAIN_UNITS is also using singleton since a game only needs one main data structure to hold all game units. The main data structure can be reused. It can be reset if mission fails or can continue the game if mission accomplishes without create a new instance of the class.

Factory Method Pattern

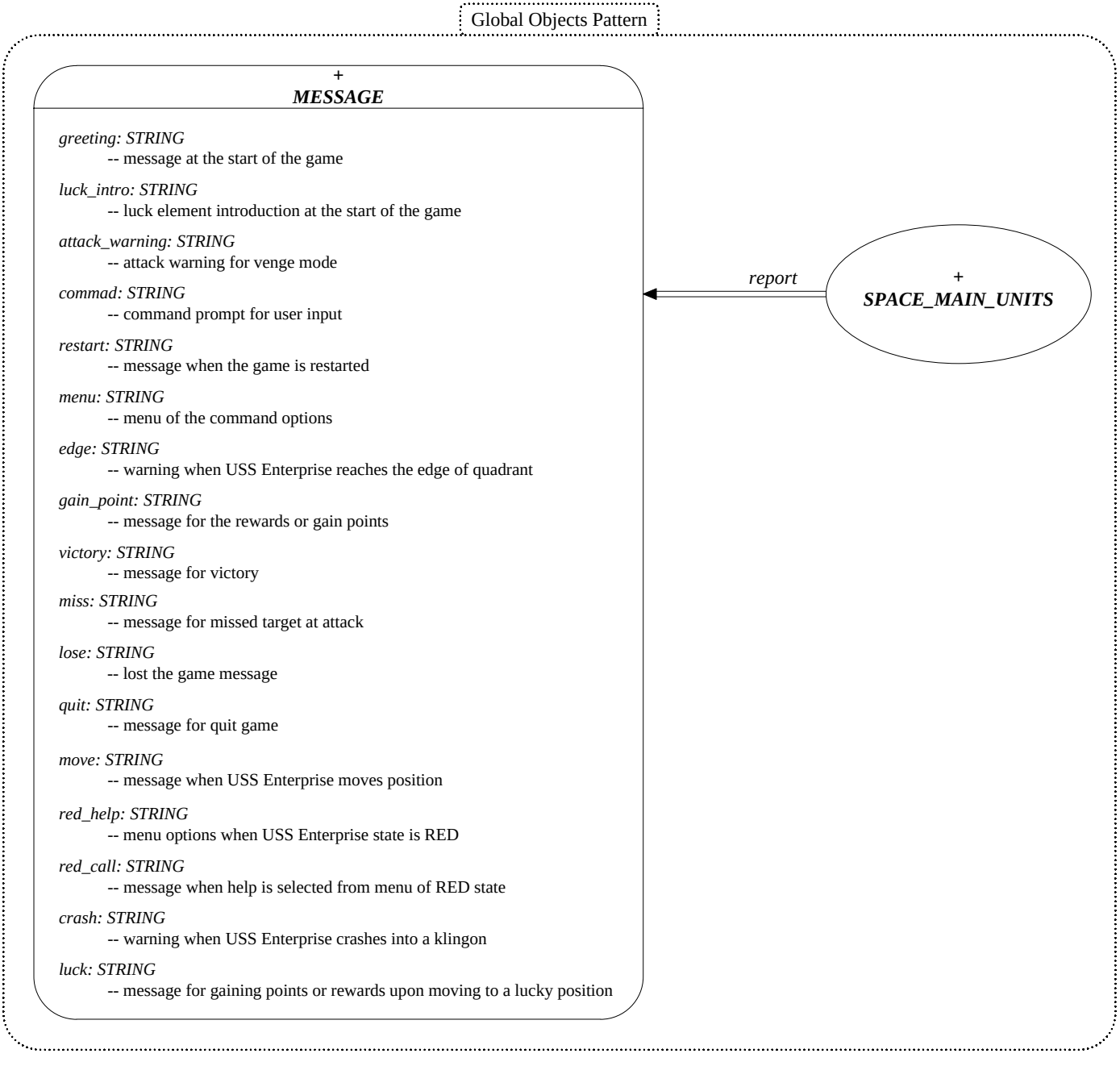


Factory Method Pattern

As the application design breaks down further after MVC pattern, it turns to how game units can be designed and implemented. A game needs several units to form roles or objects for the story. As the game grows, more units are added. Abstract factory pattern and factory method pattern are the potential candidates for this specification. Abstract factory pattern provides an interface for creating families of related objects without specifying their concrete classes. Factory method pattern defines an interface for creating an object but let subclasses decide the specific class to instantiate. For all different types of game units, they are more independent than dependent on each other. In the story of this game, the game units are in different categories instead of families. Therefore, factory method pattern is the better candidate on implementing the game units.

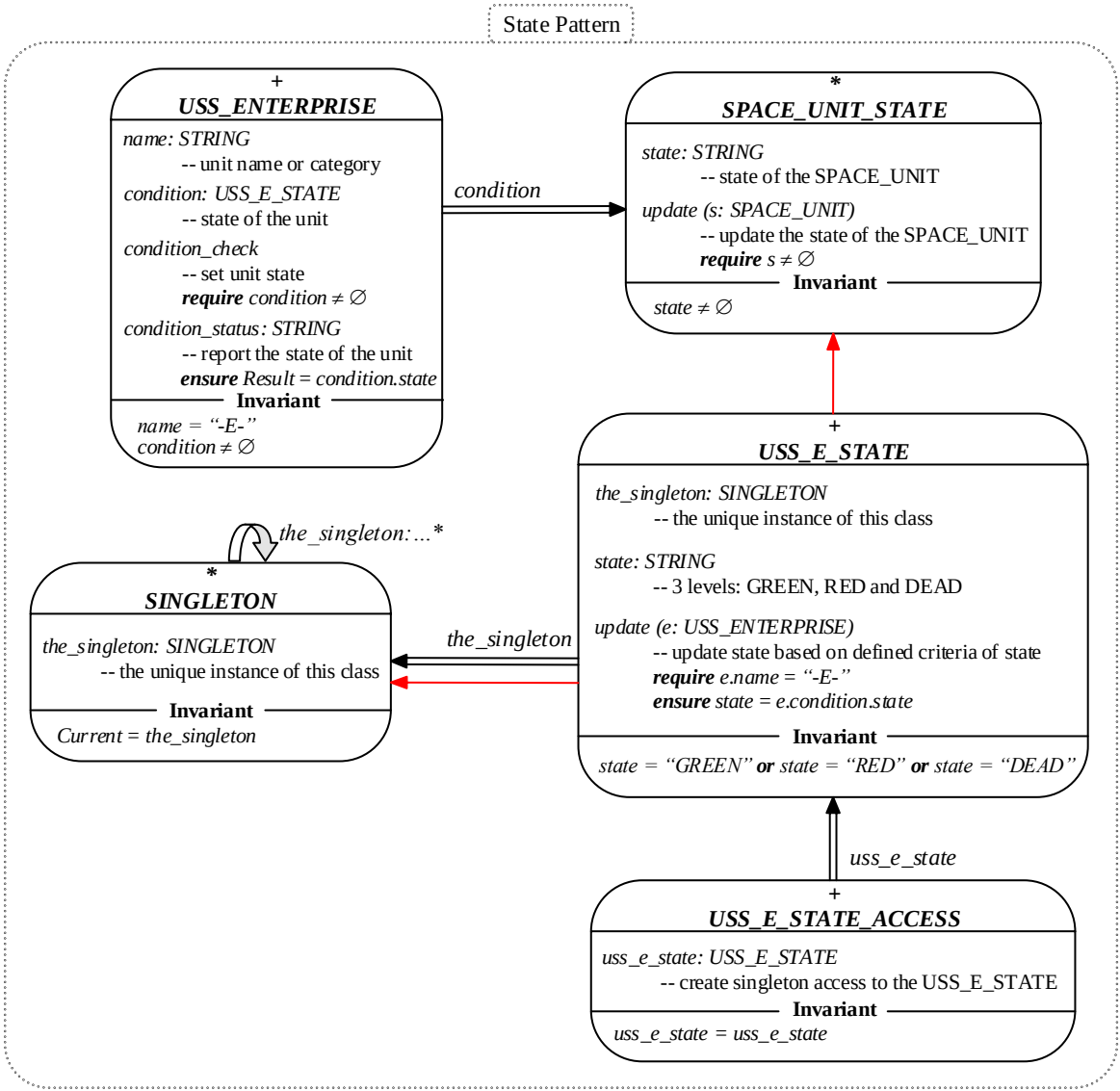
The final version of the game unit has attribute “id” and feature “set_id”. Attribute “id” simply stores the index of the unit from the list. Initially, two attributes “x” and “y” and feature “set_xy” were used in the unit. Attribute “x” and “y” represent the coordinate of the unit in the quadrant. When the unit is loaded to the game, it demands modular calculation to get “x” and “y” based on its index of the list. As the unit moves to a new position, modular calculation is needed again to get new “x” and “y”. Nonetheless, the advantage is each unit can see other units on the same row “x” and column “y” instantly.

To the contrary, using only attribute “id” saves the two calculations since “id” is the index of the list. But it still requires modular calculation when the unit tries to locate other units on its row or column. As a unit moves or attacks others, locating others is executed first. It may seem that “id” does not save the calculation times. However, it reduces the chance of error on attribute calculation and assignment as well as greatly saves troubleshooting time on the variety of units by isolating the problem to the only modular calculation feature. Not to mentation that it saves memory space per unit by using one attribute versus two attributes in a unit.


Global Objects Pattern
+
MESSAGE
greeting: STRING
-- message at the start of the game
luck_intro: STRING
-- luck element introduction at the start of the game
attack_warning: STRING
-- attack warning for venge mode
commad: STRING
-- command prompt for user input
restart: STRING
-- message when the game is restarted
menu: STRING
-- menu of the command options
edge: STRING
-- warning when USS Enterprise reaches the edge of quadrant
gain_point: STRING
-- message for the rewards or gain points
victory: STRING
-- message for victory
miss: STRING
-- message for missed target at attack
lose: STRING
-- lost the game message
quit: STRING
-- message for quit game
move: STRING
-- message when USS Enterprise moves position
red_help: STRING
-- menu options when USS Enterprise state is RED
red_call: STRING
-- message when help is selected from menu of RED state
crash: STRING
-- warning when USS Enterprise crashes into a klingon
luck: STRING
-- message for gaining points or rewards upon moving to a lucky position
report
+
SPACE_MAIN_UNITS

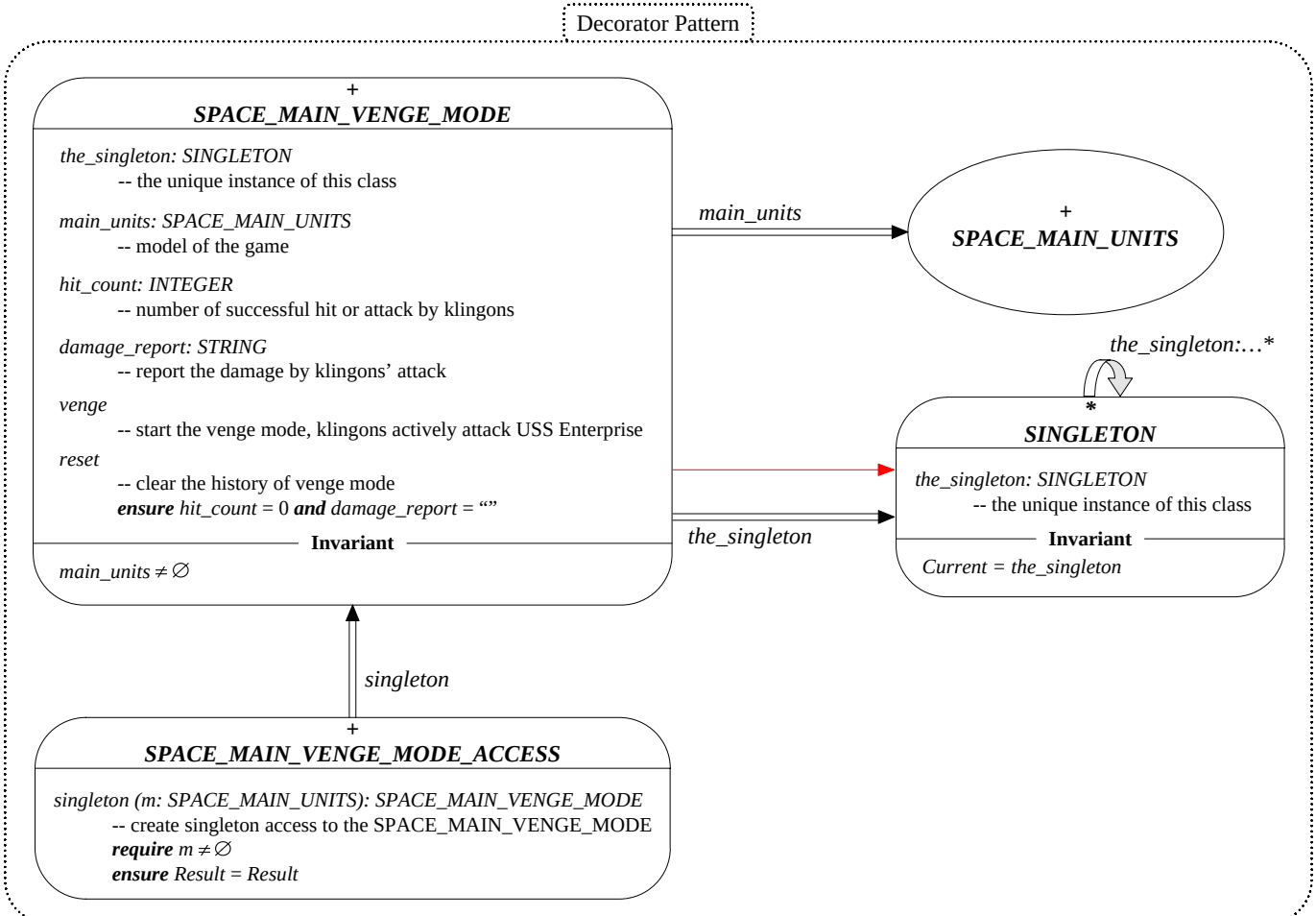
Global Objects Pattern

Text-based game literally delivers the impression that the text message is used for interaction between user and game. Some of the messages such as scores are changing dynamically while the user is playing in the game. But most of the messages are fixed constants. They are pre-defined as constants in the application. So it is reasonable to put them together as a global constant in a class. As the game grows, we may need to add more messages for the new theme. Besides, they can be reused by other related games. Global objects pattern seems like to be the only available option to choose. It largely reduces the size of the model class and is scalable to the further game development. It also helps isolate the problem for functions involved with messages.


State Pattern
+
USS_ENTERPRISE
name: STRING
-- unit name or category
condition: USS_E_STATE
-- state of the unit
condition_check
-- set unit state
require condition ≠ ∅
condition_status: STRING
-- report the state of the unit
ensure Result = condition.state
Invariant
name = "-E-"
condition ≠ ∅
condition
*
SPACE_UNIT_STATE
state: STRING
-- state of the SPACE_UNIT
update (s: SPACE_UNIT)
-- update the state of the SPACE_UNIT
require s ≠ ∅
Invariant
state ≠ ∅
+
USS_E_STATE
the_singleton: SINGLETON
-- the unique instance of this class
state: STRING
-- 3 levels: GREEN, RED and DEAD
update (e: USS_ENTERPRISE)
-- update state based on defined criteria of state
require e.name = "-E-"
ensure state = e.condition.state
Invariant
state = "GREEN" or state = "RED" or state = "DEAD"
the_singleton:...*
*
SINGLETON
the_singleton: SINGLETON
-- the unique instance of this class
Invariant
Current = the_singleton
the_singleton
uss_e_state
+
USS_E_STATE_ACCESS
uss_e_state: USS_E_STATE
-- create singleton access to the USS_E_STATE
Invariant
uss_e_state = uss_e_state

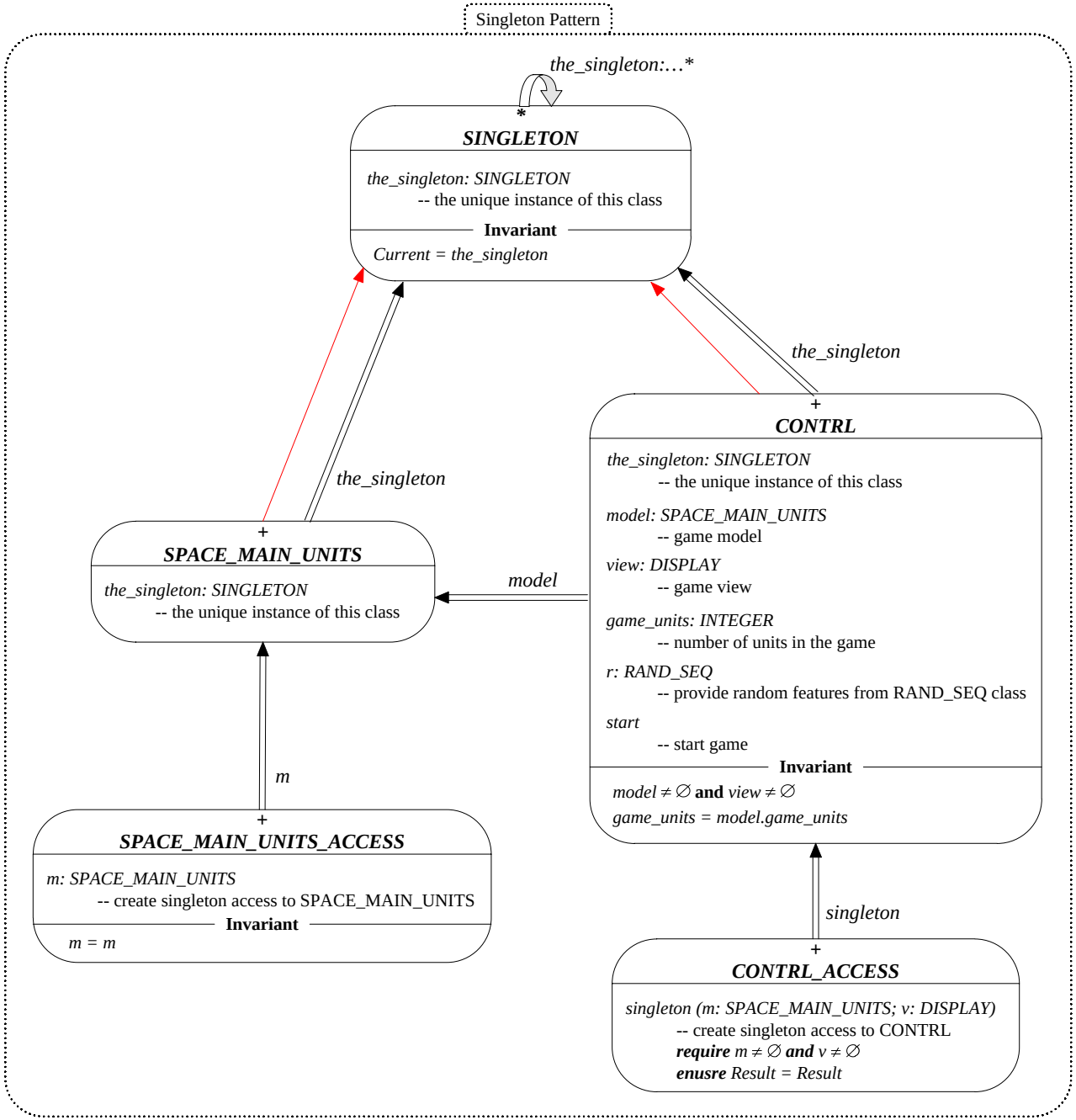
State Pattern

As the game specification, multiple states are required. Therefore, state pattern is used. The conventional state pattern is adapted a little in the design. Instead of having multiple state classes, one self-maintained state class is implemented. It provides the up-to-date state report which largely reduces the size of the condition coding such as “if… elseif…” in the main role unit. The main role unit does not need to know the state but only use attached state class to print out the current state in its feature. Meanwhile, the state class keeps updating itself according to its attached main role unit. The change of supplies triggers the change of the state. As the state changes the condition of the unit is updated accordingly and warning message and different menu option are provided to the player.


Decorator Pattern
+
SPACE_MAIN_VENGE_MODE
the_singleton: SINGLETON
-- the unique instance of this class
main_units: SPACE_MAIN_UNITS
-- model of the game
hit_count: INTEGER
-- number of successful hit or attack by klingons
damage_report: STRING
-- report the damage by klingons' attack
venge
-- start the venge mode, klingons actively attack USS Enterprise
reset
-- clear the history of venge mode
ensure hit_count = 0 and damage_report = ""
Invariant
main_units ≠ ∅
main_units
+
SPACE_MAIN_UNITS
the_singleton:…*
*
SINGLETON
the_singleton: SINGLETON
-- the unique instance of this class
Invariant
Current = the_singleton
the_singleton
singleton
+
SPACE_MAIN_VENGE_MODE_ACCESS
singleton (m: SPACE_MAIN_UNITS): SPACE_MAIN_VENGE_MODE
-- create singleton access to the SPACE_MAIN_VENGE_MODE
require m ≠ ∅
ensure Result = Result

Decorator Pattern

Decorator Pattern is added lastly after all the implementations are completed and tested successfully. Having a production game, I was considering adding more functions to the game but not to change any part of it. With or without the new functions the production game should work fine. Decorator pattern just fits into the specification. So a decorator class SPACE_MAIN_VENGE_MODE is designed and implemented. The venge mode uses existing data structure to allow Klingons to attack back Enterprise when they are under attack. This added or extended intelligence functionality creates challenge and more fun to the player and turns a simple shooting game into a strategy game.


Singleton Pattern
the_singleton:…*
*
SINGLETON
the_singleton: SINGLETON
-- the unique instance of this class
Invariant
Current = the_singleton
the_singleton
+
CONTRL
the_singleton: SINGLETON
-- the unique instance of this class
model: SPACE_MAIN_UNITS
-- game model
view: DISPLAY
-- game view
game_units: INTEGER
-- number of units in the game
r: RAND_SEQ
-- provide random features from RAND_SEQ class
start
-- start game
Invariant
model ≠ ∅ and view ≠ ∅
game_units = model.game_units
the_singleton
+
SPACE_MAIN_UNITS
the_singleton: SINGLETON
-- the unique instance of this class
model
m
+
SPACE_MAIN_UNITS_ACCESS
m: SPACE_MAIN_UNITS
-- create singleton access to SPACE_MAIN_UNITS
Invariant
m = m
singleton
+
CONTRL_ACCESS
singleton (m: SPACE_MAIN_UNITS; v: DISPLAY)
-- create singleton access to CONTRL
require m ≠ ∅ and v ≠ ∅
enusre Result = Result

Singleton Pattern

Singleton Pattern is widely used for any scenario that requires a single instance in the system. It is also the game specification. Controller class has only one instance in the system. Meanwhile, there are other classes are implemented in singleton. Therefore, multiple singleton class is implemented.

The game main structure or model is using singleton as the earlier discussion in MVC pattern section. It keeps the records of the player such as scores and state as the player continues the consecutive missions. It also saves the memory space without creating new instance and consequently shortens a new mission loading time.

In addition, decorator pattern and state pattern all incorporate the singleton pattern.