[1 Cover]

[93 – [KAPPA]

Supervisor:
Dr. Goldschmidt Balázs

Members:

[Houssam Mehdi]	[DE8DFE]	[houssam_mahdi_98@hotmail.com]
[Khouloud Khezami]	[PO9XKD]	[khouloudkhezami24@gmail.com]
[Tony Azar]	[]	[tony.azarr@gmail.com]
[Mahdi Karray]	[RO7C7Y]	[karraymahdi1@gmail.com]
[Rovshan Sirinli]	[I5XW3M]	[rovsensirinli5@gmail.com]

[19/02/2018]

2. Requirements, project, functionality

2.1 Requirement definition

The requirements for this project will be divided into 4 parts, as follow:

1. *User: Control the workers, move the boxes using the workers movement.*
2. *Functional: Add boxes, holes and switches.*
3. *Non-functional: Allow up to multiple users to control the workers, one each.*
4. *Implementation: Must run on all Java platforms.*

- *The boxes should be able to be moved in 4 directions: right, left, up and down. Except if there's a wall blocking the way.*
- *The switches must be able to be turned on/off by moving the boxes on them and should not be controlled by the user alone.*
- *The holes should be triggered by the switches and its status should be on/off according to the switch sign. When the switch pointing to a hole is turned off, the hole acts as a normal block, and when it's on it should be a hole again.*
- *Whenever a box is placed on an active hole, the box should disappear, and whenever a worker is placed on an active hole he dies.*
- *The player should be able to finish the game by placing all the boxes to their designated places, and the player who moved the most wins. The game also finishes if the boxes cannot be moved anymore.*

2.2 Project plan

Our team uses the agile approach of working which means that everyone knows about everything, which will make it easier to adapt to requirements changes during the development process.
The schedule is flexible depending on how much work we have to get done, as well as the deadline.
The sub team will be constituted as follows, two persons for development, two for testing, and one for supporting, one person will also be responsible of marketing and finance, and finally one responsible for internal and externa communication.

2.3 Problem description

In a warehouse boxes are stored. The floor of the warehouse is divided into equal squares. Each box occupies a single square. The boxes can be pushed by the workers, but after a move a box always stands on a single square.

There are multiple workers at the warehouse. Their task is to move the boxes to their designated place. The warehouse has walls and pillars, which obstruct the move. The boxes, when pushed, can push other boxes in their way. When a box is pushed on a worker, the worker is accordingly pushed to the adjacent square as well. If the worker is nowhere to be pushed (e.g. there is a wall), the worker dies. The boxes' size cannot change.

The floor also contains holes. If a box is pushed onto a hole, the box falls in the hole (disappears). If a worker steps into a hole, the worker dies. Some holes have a switch somewhere else. These holes work as regular squares when switched off, and as holes when switched on. The switches are like regular squares, they are switched on only when a box is on them. The worker standing on a switch does not open the corresponding hole.
The game ends when either all boxes are at their designated place, or there are no more boxes that can be pushed. Whoever pushed the most boxes to their designated place, wins.

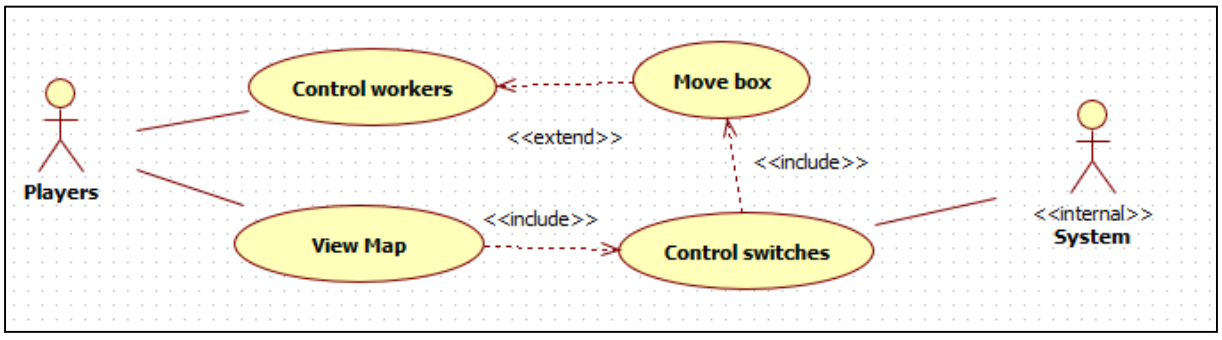
2.4 Glossary

- *Actor: an actor can be a human user, or a hardware or any other entity that would represent a concrete role with the system. In other words an actor interacts with a sub-system of the modelled system. In this case, the actors are the Player and the System.*
- *Extends: is a relationship between two cases, for instance use-case A extends use-case B , this means that A contains all the steps of B with further substeps and options.*
- *Include: is a relationship between two cases, for instance if use-case A includes use-case B, this means that B is part of A, an example would be the following, use-case : "take a trip" the take a trip includes "choose destination" and "fuel the car".*

2.5 Essential use-cases

We chose 4 use cases to describe the functionality of the product. Control workers use case is responsible for moving the workers which may result in push of boxes which is described by the use case Move box. The System is responsible for controlling the switches which uses the movement of boxes at some point therefore includes the Move box use case. The View map use case draws the position of the objects (workers, boxes, walls, holes).
The map includes the squares on which the game is played.

2.5.1 Use-case diagram



2.5.2 Use-case descriptions

Use-case name	*Control workers*
Short textual description	*The players can move workers in the map*
Actors	*Players*
Dialog, scenario	*1. The players moves the workers left, right, up or down*

Alternate scenario	*1.A.1. When the worker moves towards a box, a box is moved accordingly*
Alternate scenario	*1.B.1. If a worker is between a wall and a box and the box is pushed on by another worker, he dies.*
Alternate scenario	*1.C.1.if a worker moves or is moved onto a hole, he dies.*
Alternate scenario	*1.D.1. if all workers are dead, the game ends.*

Use-case name	*View Map*
Short textual description	*The player views the map.*
Actors	*Players*
Dialog, scenario	*1. The system draws the current state of the map.* *2. The player observes the current state of the map.*

Use-case name	*Move box*
Short textual description	*The box can be pushed by a worker.*
Actors	*Players*
Dialog, scenario	*1.If a worker moves towards a box and there is no wall in the direction of the push, the box is moved to the next square.*
Alternate scenario	*1.A.1.If a box has two walls or has another box which is adjacent to wall adjacent to it, the box becomes immovable.*
Alternate scenario	*1.A.1.A.1 if all the remaining boxes in the map are immovable, the game ends.*
Alternate scenario	*1.B.1. if a box is pushed towards a hole, it disappears.*

Use-case name	*Control switches*
Short textual description	*A switch is a regular square and is linked to a hole. The switches can be on or off.*
Actors	*System*
Dialog, scenario	*1. if the box is on the switch, the hole works as a hole.* *2. if the box is not on top of the switch These holes work as regular squares when switched off.*

2.6 Protocol

Start (date & time)	Duration (hours)	Performer(s) name	Activity description
14/02/2018 17:00	*1.5*	*Houssam Mehdi* *Khouloud Khezami* *Mahdi Karray* *Tony Azar* *Rovshan Sirinli*	*Meeting.*
17/02/2018 21:00	*3*	*Houssam Mehdi*	*Requirements 2.1, 2.2 and 2.3 and filled the complete document*
17/02/2018 13:00	*2*	*Mahdi Karray*	*Use-case 2.5.1 and 2.5.2*
17/02/2018	*2*	*Khouloud Khezami*	*Requirements 2.2, Glossary*
17/02/2018 21:00	*2*	*Tony Azar*	*Requirements 2.3, Glossary*

17/02/2018 13:00	*2*	*Rovshan Sirinli*	*Use-case 2.5.0 and 2.5.2*

3. Analysis model – version 1

3.1 Object catalog

Fields,Map,Thing,Game,Player,Worker,DesignatedSquare,Wall,Hole,Switch,Box

3.1.1 Fields

Fields represent the squares in the task description the other interactive elements of the game will be associated with these and in some way placed on them their responsibility is to be able to get info about neighboring fields and either acccepting or removing objects from themselves .It is used by the map as grounds for other objects.Every thing element must have one and uses it at almost every instant to fulfill its requirements .It plays a quite important role as the grounds for everything.It contains at most 2 Thing objects which through the getter and setters of a field one thing can acces the other.

3.1.2 Map

Map is a collection of Fields and is responsible for placing things and positioning them in some fields according to the specifications.

3.1.3 Thing

Describes any object that could be placed on the Field is used as a common way to acces or reference some other objects .

3.1.4 Player

Each person ,external actor is able to have one and is used as a connection between the game and the outside world connects with objects of Worker class.Has the responsibility of assigning itself toa worker in game component.is connected to the DesignatedSquare which determines the abount of points a player gets mid game.

3.1.5 Worker

Represents the in game player main object from the users point of view is the only object controlled by outside actors .has the responsibility of making movements initiating movements which might affect other objects of class Box.Handles when BeingSteppedOnBy by other objects in a unique way.

3.1.6 DesignatedSquare

Represents the positions which the actor has to push other objects to is the one of the main objectives of the game also,is responsible for checking the state of the object it shares the field with also determines how player recive or lose points. Handles when BeingSteppedOnBy by other objects in a unique way.

3.1.7 Wall

Repserents pillars or walls in the description .Cannot be moved or dislocated in any way once placed in the game's beginning.Handles when BeingSteppedOnBy by other objects in a unique way.

3.1.8 Switch

It is a entity that asoociates with a field some holes can have this entity as an assotiation .if it is turned on the hole will work otherwise the associated hole will function as a normal field .Only when BeingSteppedOnBy with a Box entity this entity will change its state also it is through the getters and setters we are able to modify the corresponding hole.

3.1.9 Box

Repsesents a box in the description can be BeingSteppedOnBy by other boxes and workers .It handles its own movement BeingSteppedOnBy in a way that if any enity pushes this it will just pass on the push to the same direction .Can be destroyed or removed from its Field.

3.1.10 Hole

Has an inner state variable by default it is on but if associated with a Switch it is off in the beginning if the inner state of the corresponding Switch has to change for this parameter to change.When BeingSteppedOnBy by other objects requests from the field to remove it .

3.1.11 Game

Handles the overall processes of game strat and end also number of players Also has a Map entity .It does not have many respnsibilities .Might be as a ground for custom map editing or Level management for later design ,this is not yet implemented .

3.2 Class description

One of the main entities is Thing which is an abstract class. A Thing is anything that can be placed on the squares of the floor.

A Thing can be be stepped on by a woker or a a box, this behaviour is haddeled by beingSteppedOnBy(b:box) method and beingSteppedOnBy(w:worker) which are purely virtual methods and will be implemented by its childeren with different implementation in each.

Thing is generalization of the following entities: Worker, box, hole, switch, walls(we decided that pillars and walls are the same) and designatedPlace.

We deciced that the best way to resolve the problem is by considering everthing an object that can be place on squares(fields)

Fields in other terms square is another main entity: it can accept a Thing on it as well as removing it, it can also get its neighbors and returns a pointer referencing the Thing on it. On a field we can have either 0, 1 or 2 Things on it, the class implements a method checkNull, that return a bool and tells whether there is a thing on the field or not.

3.2.1 Box

- **Responsibility**

A box can move to a given direction. A box disappears when it gets into a hole. Remenberes a reference to the object(Thing) which pushed it.

- **Superclasses**

Thing

- **Interfaces**

-

- **Attributes**

Public Thing previous This attribute is a reference to the Thing which may call this object by the **HitByBox** or **HitByWorker** ,it is by default Null and will only change while the **HitByBox** and **HitByWorker** is active for the current worker(this).It may be asked by the **GetPusher(t: Thing)** of the **DesignatedSquare** when adding points to the player **.**

- **Methods**

  - **Public void die() :** the box gets destroyed .It can only be called by a hole when the box falls onto the hole.
  - **Public bool HitByWorker (w:Worker, d:Direction) :** The current object(this box) is asked to relocate by the other object of type Worker passed as a paramater **w** .The current box will ask the object(neigbour) in the direction of the push **d** with the same function but it will pass itslef as the box parameter of the **HitBtBox(this,d)** if succesful (the asked object will return true) then will relocate ,Change its **previous :Thing** parameter to the passed object **w** and fianlly return **true** if not then will reamin in the same field and return **false.**

  - **Public bool HitByBox (b:Box, d:Direction)** The current object(this box) is asked to relocate by the other object of type Box passed as a paramater **b** .The current box will ask the object(neigbour) in the direction of the push **d** with the same function but it will pass itslef as the box parameter of the **HitBtBox(this,d)** if succesful (the asked object will return true) then will relocate ,Change its **previous :Thing** parameter to the passed object **w** and fianlly return **true** if not then will reamin in the same field and return **false.**

3.2.2 DesignatedSquare

- **Responsibility**

Adding points to the player whose worker delivered a box to a designated place.

Substracts points to the player whose worker moves a box that reached a designated place.It is also capable of determining which player pushed first in a specific push given a direction.

- **Superclasses**

  Thing

- **Attributes**

- **Methods**

  - **public void addPoints(p:Player):** when a worker gets a box to a designated place, this method will be executed to add points to its corresponding player, the player passed by parameters.
  - **Public void removePoint(p:Player):** when a worker moves a box from a designated place, its corresponding player will lose points, and this method handles the substruction of points.
  - **Public bool checkBox(b:Box):** check if a box is already on the designated place
  - **Public Thing GetPusher(t: Thing):** This method is used to get a reference to the object(thing) which made the last push to the asked object.When a box somehow moves onto a DesignatedSquare the this method will ask the box for its's Box's **previous** pusher which will return another Thing type reference(can be worker or box does not matter) then we will again do the same prosedure as long as we dont get a Null .The worker by definition of the requirements is the object of type thing which started the push which means that the first object that returns Null for its **previous** is the first worker who started the push.This function will be used whenever determining which player we need to add to or remove points from is requested.

3.2.3 Direction

- **Responsibility**

This enumeration lists the possible neighboring directions of Fields: Up, Down, left or right.

And also the direction a worker can take: Up, Down, left or right

3.2.4 Fields

- **Responsibility**

Stores Things that can appear in the Warehouse. Fields knows its neighbours.Represents the Squares in the floor of the warehouse.

- **Superclasses**

  -

- **Interfaces**

  -

- **Attributes**
  - Neighbors[Direction]:stores the neighbouring fields in the possible directions
  - -thing:Thing[0..2]: Things currently on the Field

- **Methods**
  - **Public void Accept(t:Thing):** accept the t:Thing on the current field using its getters and setters .
  - **Public void remove(t:Thing):** remove the t:Thing from the current field field using its getters and setters
  - **Public Fields getNeighbour(f1:Fields, d:Direction):** returns the Fields according to the given direction
  - **Public void setField(d:Direction, f:Fields):** Given a field sets a field as a neighboring field .
  - **Public bool checkNull():** tells if there is something on the field or not if the field is empty returnes true false otherwise
  - **Public Thing GetThing(i: int)** returnes the Thing from the list of things on the field int **i** will show the
  - **Public SetThing(i: int, t: Thing)** sets the thing in the index i which can not be more than 2 with the passed Thing .

3.2.5 Game

- **Responsibility**

  Manages the game

- **Superclasses**

  -

- **Interfaces**

  -

- **Attributes**

  -

- **Methods**
  - **Public void addPlayer(p:Player):** adds player to the game
  - **Public void startGame():** starts the game
  - **Public void endGame():** ends the game

3.2.6 Hole

- **Responsibility**

If a worker or a box step on a the hole, so it dies.

- **Superclasses**

Thing

- **Interfaces**

-

- **Attributes**
  - + State :Bool : it give the state of the hole (so if so the switch is on so it's a empty field if not it's a hole)

- **Methods**
  - -

3.2.7 Map

- **Responsibility**

  Stores the Fields and maintains the designated places

- **Superclasses**

  Thing

- **Interfaces**

  -

- **Attributes**

  -

- **Methods**
  - **Public void addBox(b:Box):** adds a box to the grid
  - **Public void addSwitch(s:Switch):** adds a switch to the grid
  - **Public void addWorker(w:worker):** adds a worker to the grid
  - **Public void addHole(h:Hole):** adds a hole to the grid
  - **Public void addWall(w:wall):** adds a wall to the grid
  - **Public void addDesignatedSquare(d:DesignatedSquare):** adds a designatedSquare to the grid

3.2.8 Thing

- **Responsibility**

  Represents a general thing (e.g.wall, hole,switch,box,worker, DesignatedSquare) which can be on a Field.

- **Superclasses**

  -

- **Interfaces**

  -

- **Attributes**
  -Filed:Fileds → The Field the thing is on.

- **Methods**
  - HitByBox(b: Box, d: Direction): boolean
    - Indicates a push which is being made by Box **b** in direction **d** The result is determined in each case differently as per each subclass of thing .Look at a specific subclass for detailed explanation.
  - HitByWorker(w: Worker, d: Direction): boolean
    - Indicates a push which is being made by Worker **w** in direction **d** The result is determined in each case differently as per each subclass of thing .Look at a specific subclass for detailed explanation.
  - GetField(): Fields
    - Return the field of the Thing . Which the object(Thing) is standing on , associated with.

  - SetField(): Fields
    - Set the field of the Thing .Which the object(Thing) is standing on , associated with.

3.2.9 Switch

- **Responsibility**

Make a hole as a empty filed or remaining as a hole

- **Superclasses**

Thing

- **Interfaces**

-

- **Attributes**
  - +State: Bool (ON or OFF)

- **Methods**
  -

3.2.10 Wall:

- **Responsibility**
  It is placed on a Fields, it stops the movement of a box and a worker.

- **Superclasses**
  Thing

- **Interfaces**

  -

- **Attributes**

  -

- **Methods**
  - **Public bool HitByWorker (w:Worker, d:Direction)** When a worker **w** tries to move on the field which has this wall this function will be called and this will always return false because the wall (because of the description) is unable to relocate the original position appointed by the map can never change midts the game.
  - **Public bool HitByBox (b:Box, d:Direction)** When a box **b** tries to move on the field which has this wall this function will be called and this will always return false because the wall (because of the description) is unable to relocate the original position appointed by the map can never change midts the game.

3.2.11 Worker

- **Responsibility**

Can move along the grid with a given direction, can push a sequence of boxes into a given direction. Dies if it gets pushed to a wall or to a hole.Remenberes a reference to the object(Thing) which pushed it.

- **Superclasses**

Thing

- **Interfaces**

  -

- **Attributes**

Public Thing previous This attribute is a reference to the Thing which may call this object by the **HitByBox** is by default Null and will change while the **HitByBox** is active for the current worker(this).It may be asked by the **GetPusher(t: Thing)** of the **DesignatedSquare** when adding points to the player **.**

- **Methods**
  - **Public void die() :** the worker gets destroyed thus the game will end for its corresponding player
  - **Public void move(d:direction):** moves the worker along the given direction The direction is specified by the Player in controll of the worker.
  - **Public bool HitByWorker(w1:worker, d: Direction):** The current worker is being pushed by another worker which will result in no change for this worker.Because when relocation does not happen the return value is always false indicating to the caller that it cannot reallocate here.
    Public bool HitByBox(b: Box, d: Direction): The current worker (this) is being pushed by the box passed in the parameters **b**.It will return a boolean value depending on the succes of the relocation of worker if worker is able to move then it will return

true if the worker is unable to relocate in the given (passed) direction then he will die and return false.

3.2.12 Direction

- **Responsibility**
  Enumeration
- **Superclasses**
  -
- **-Interfaces**
  -
- **Attributes**
  - Right
  - Left
  - Up
  - Down
- **Methods**

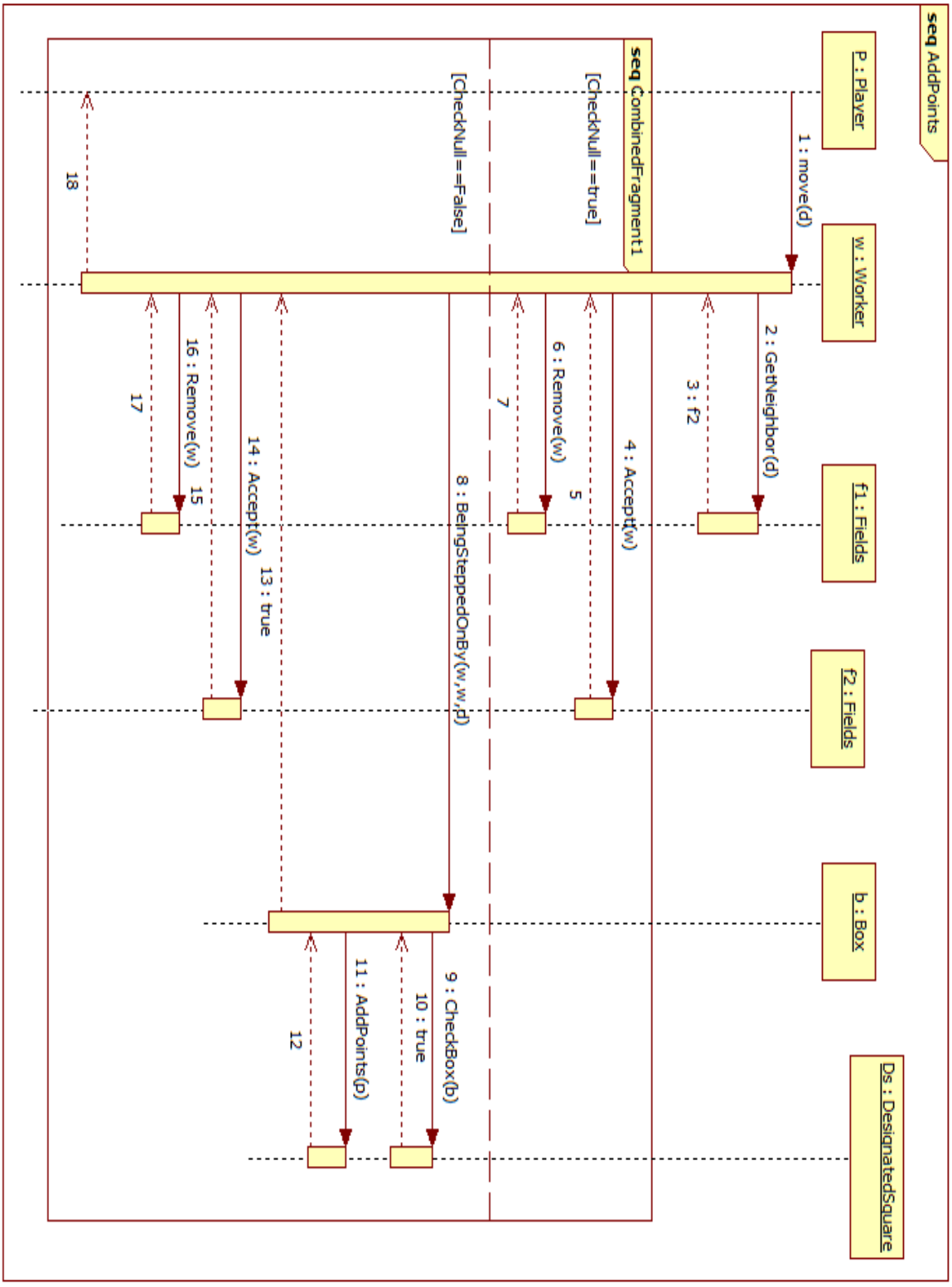
  -

3.3 Static structure diagrams

3.4 Sequence diagrams

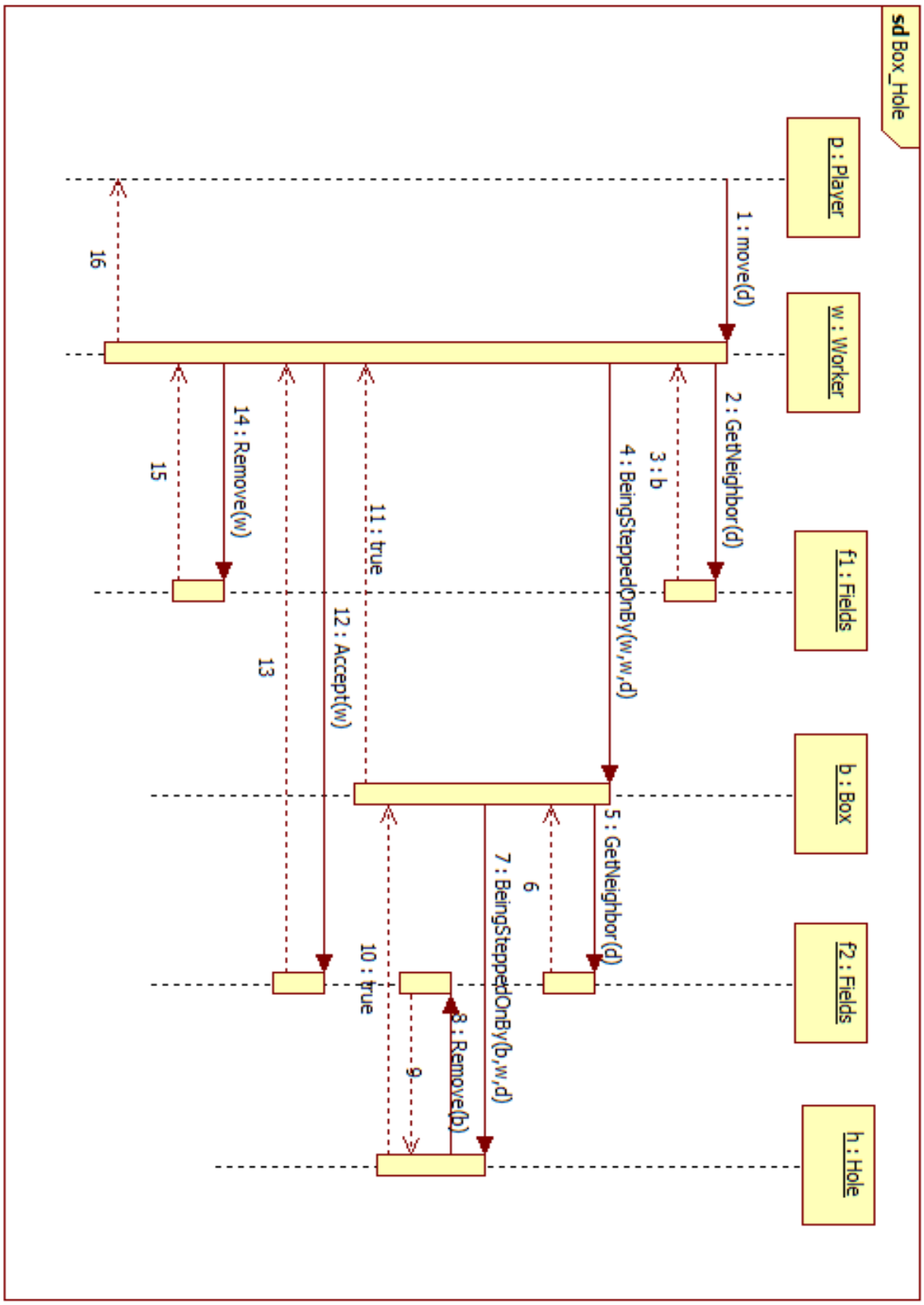
3.4.1 AddPoint

This Sequence diagram shows how to add points to the player when he push a box on a designated square.



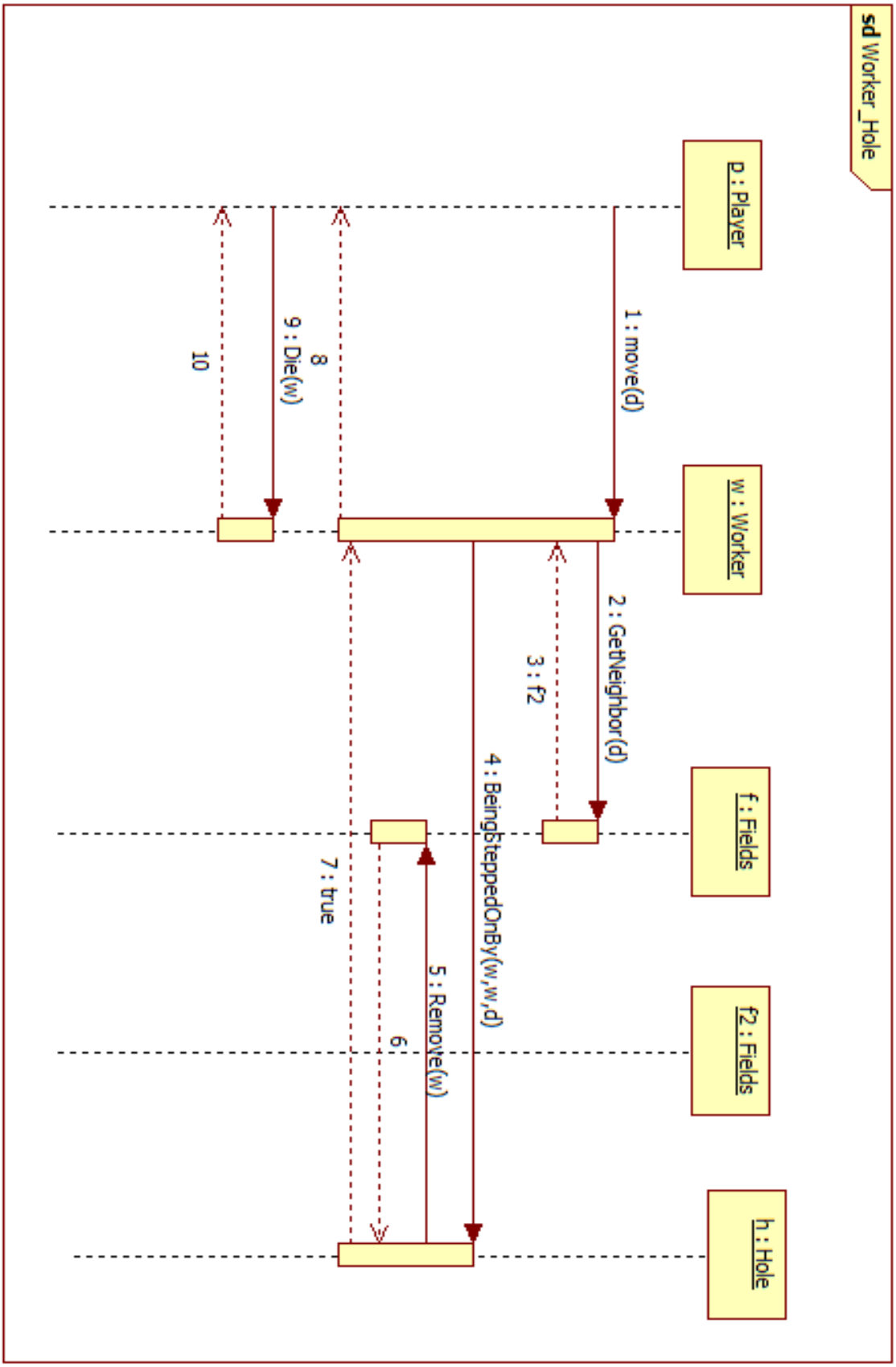
3.4.2 Box_Hole

This Sequence diagram shows when a worker push box and after the box there is a hole



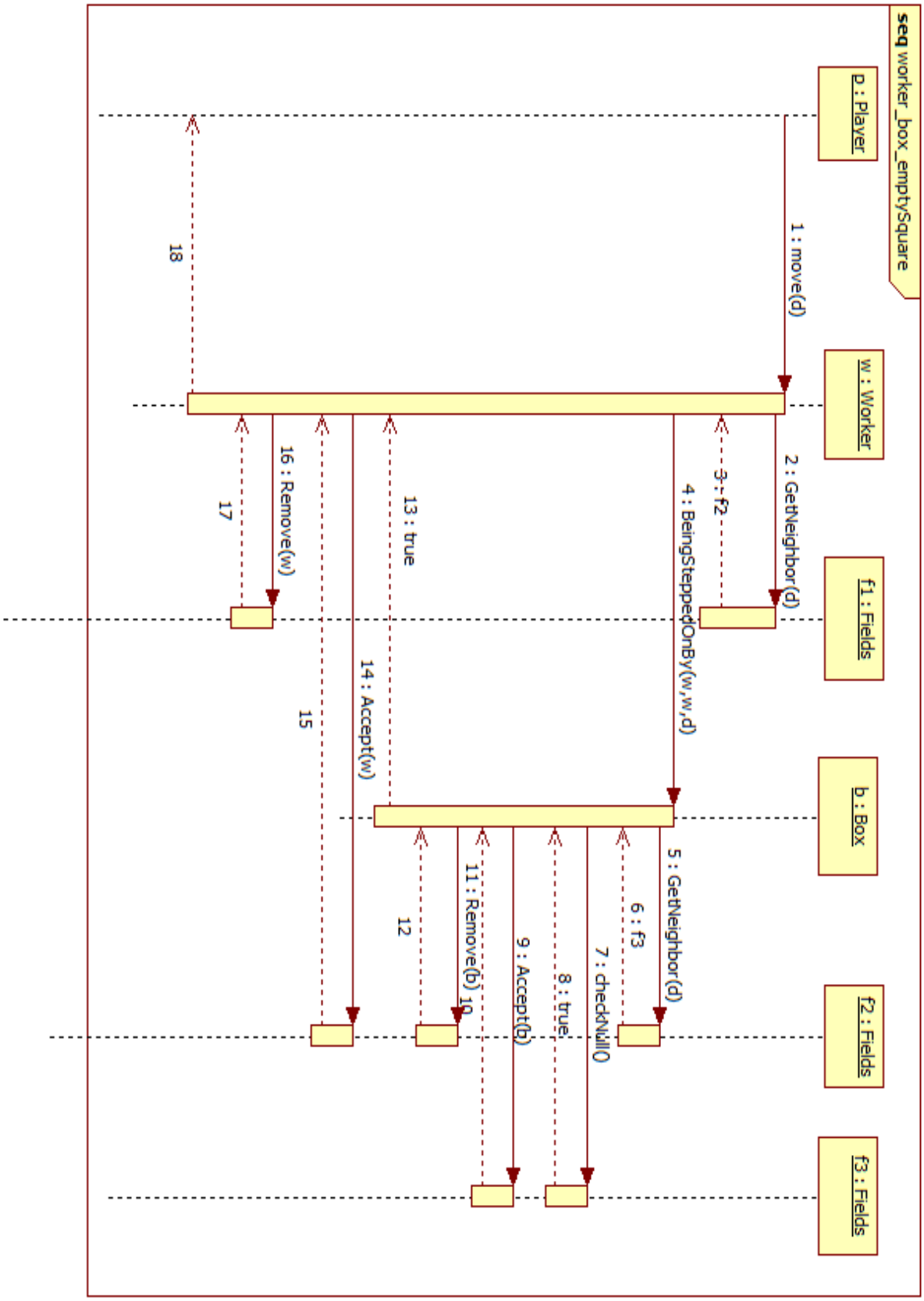
3.4.3 Worker_Hole

This Sequence diagram shows when a worker moves to a hole.



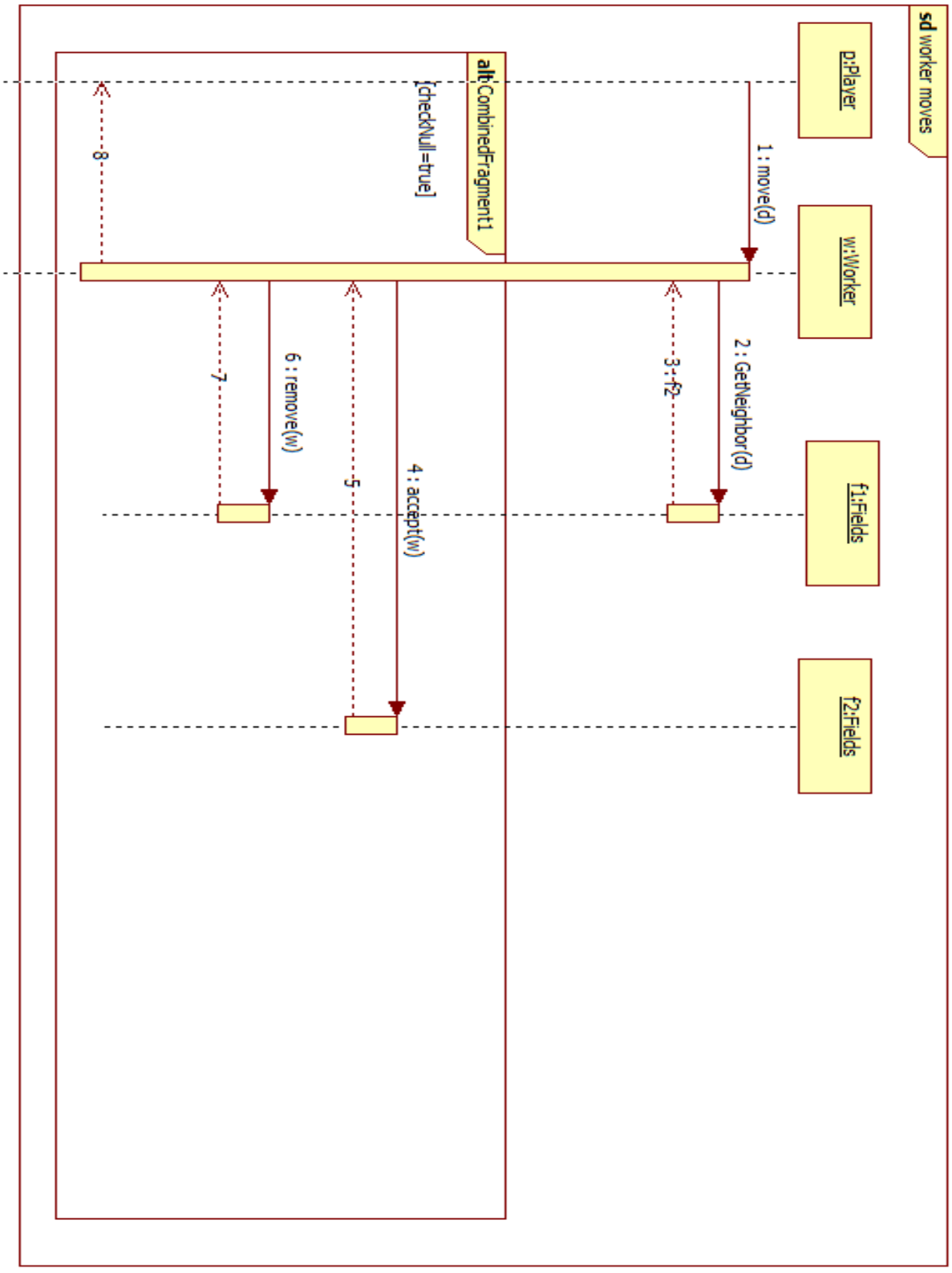
3.4.4 worker_box_emptySquare

This Sequence diagram shows how a worker moves the box to an empty field

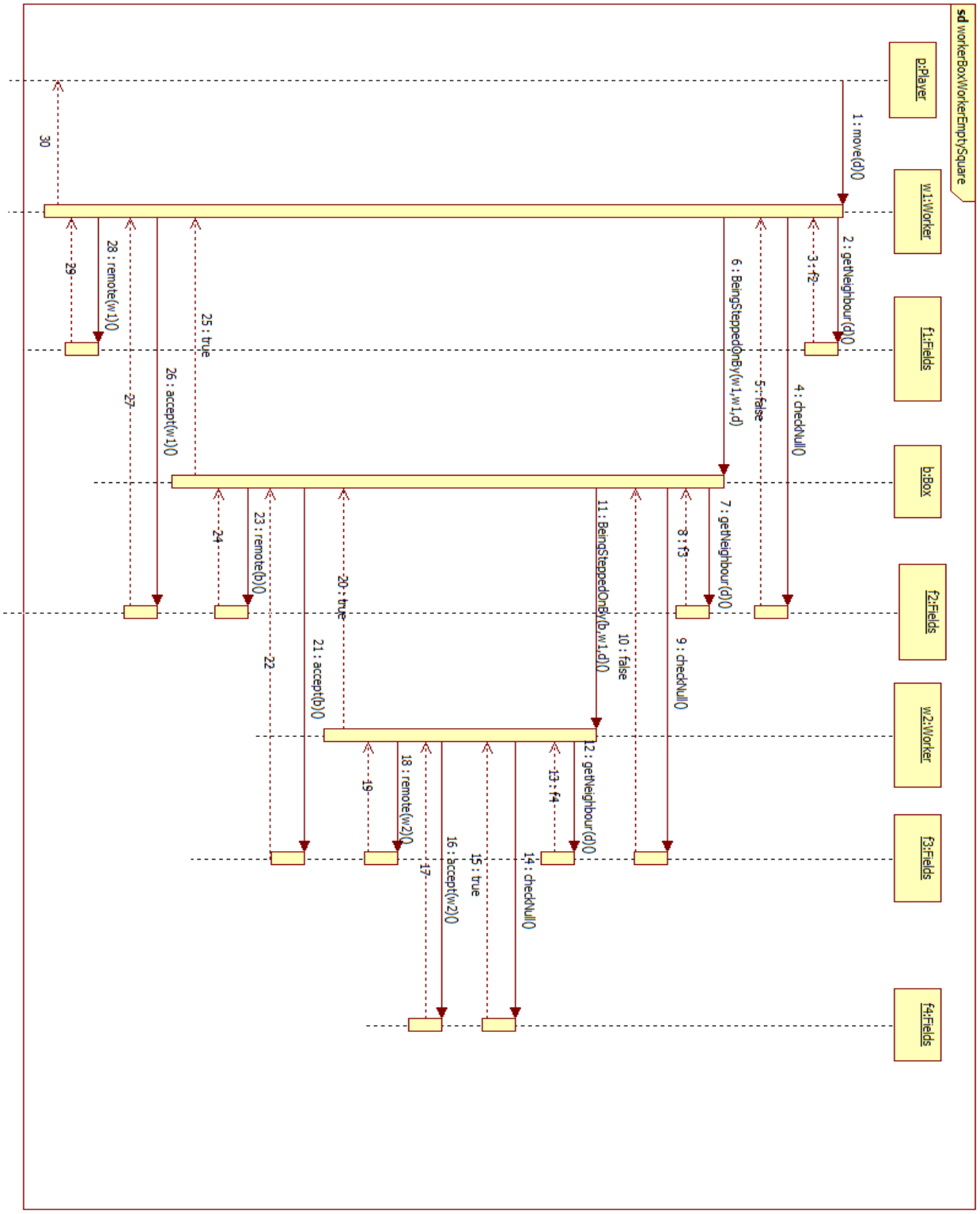


3.4.5 Worker move

This Sequence diagram shows how a worker moves into a give direction

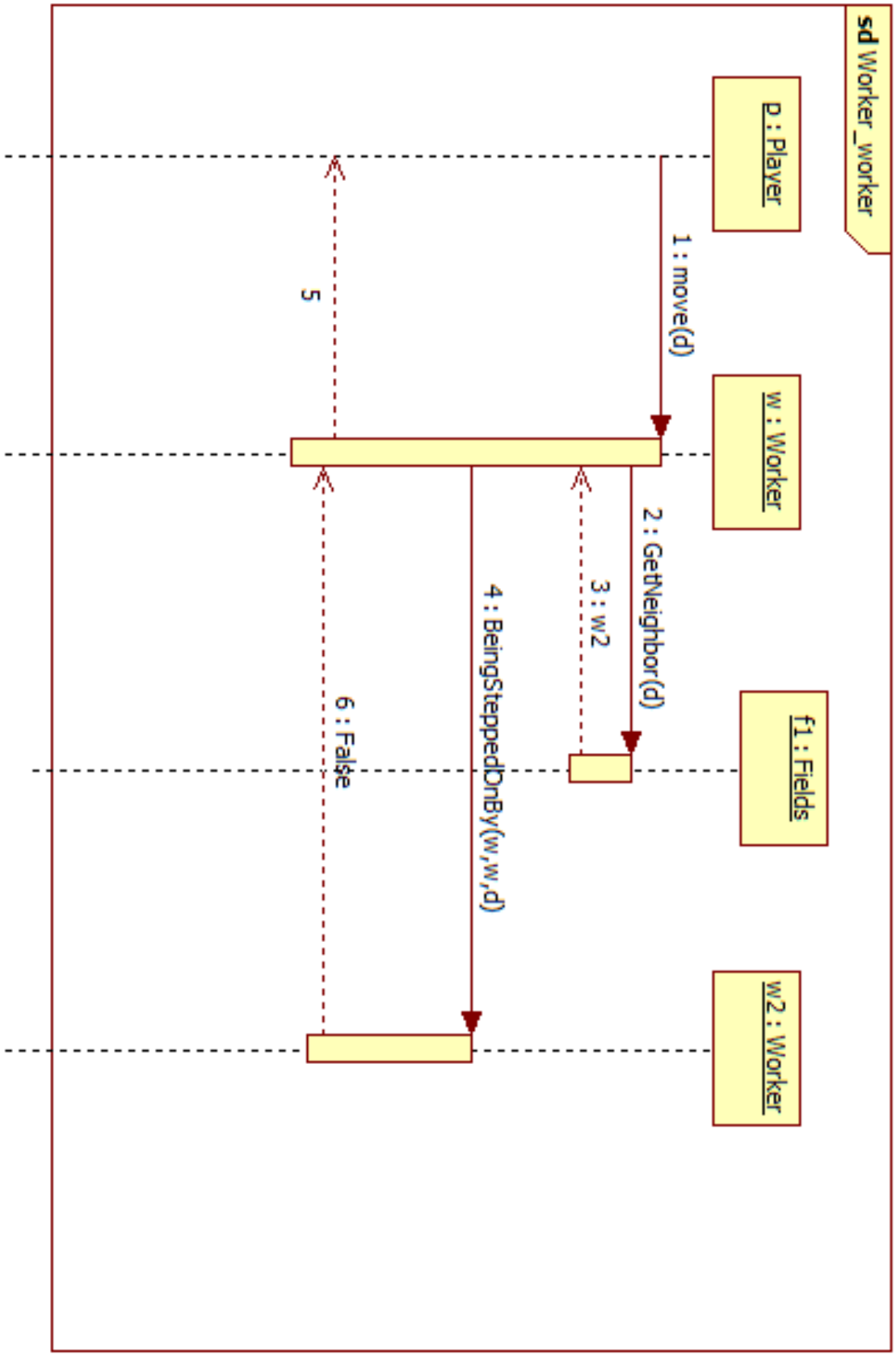


3.4.6 Worker1 Box Worker2 EmptyField



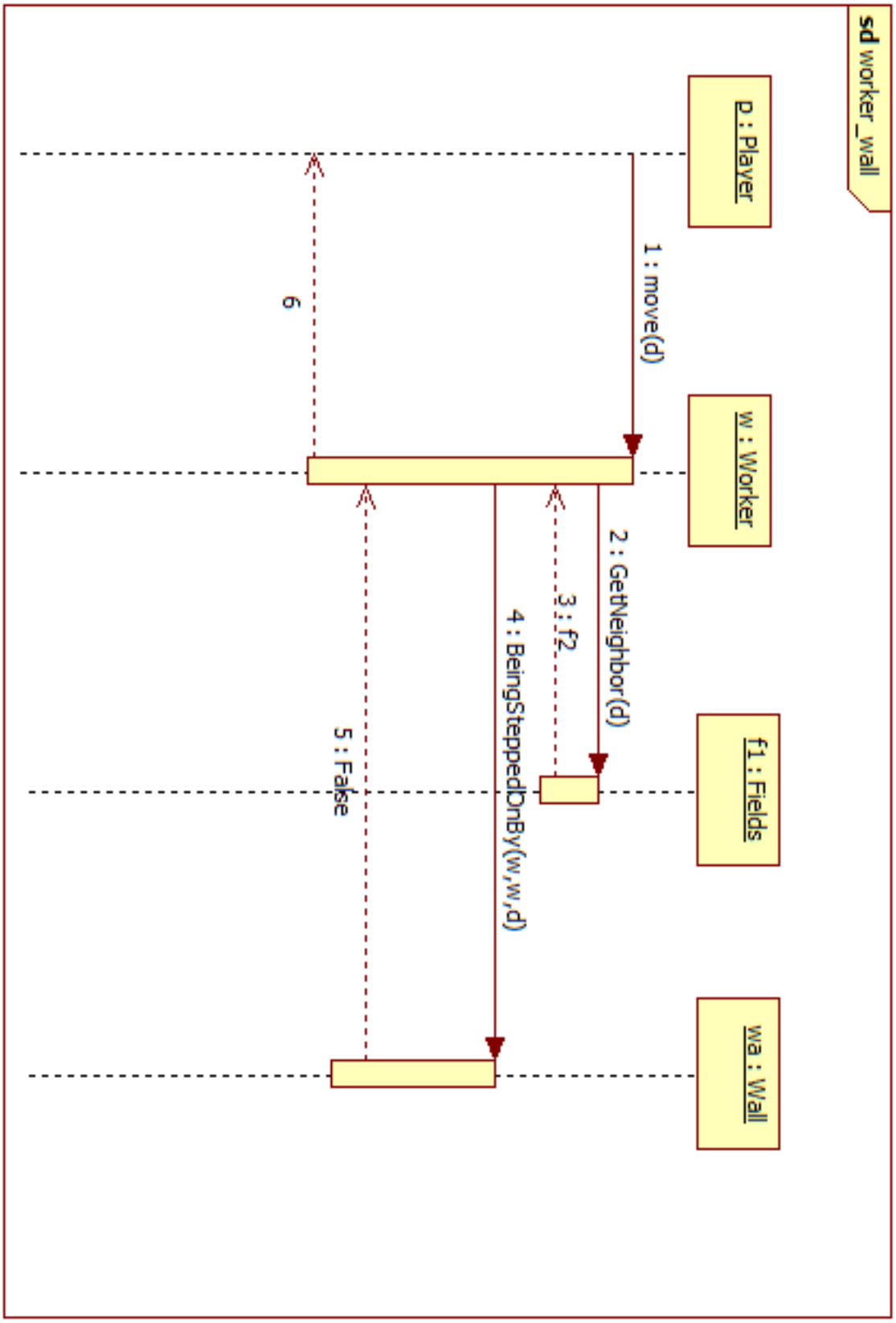
3.4.7 worker_worker

This Sequence diagram shows how a worker push a worker



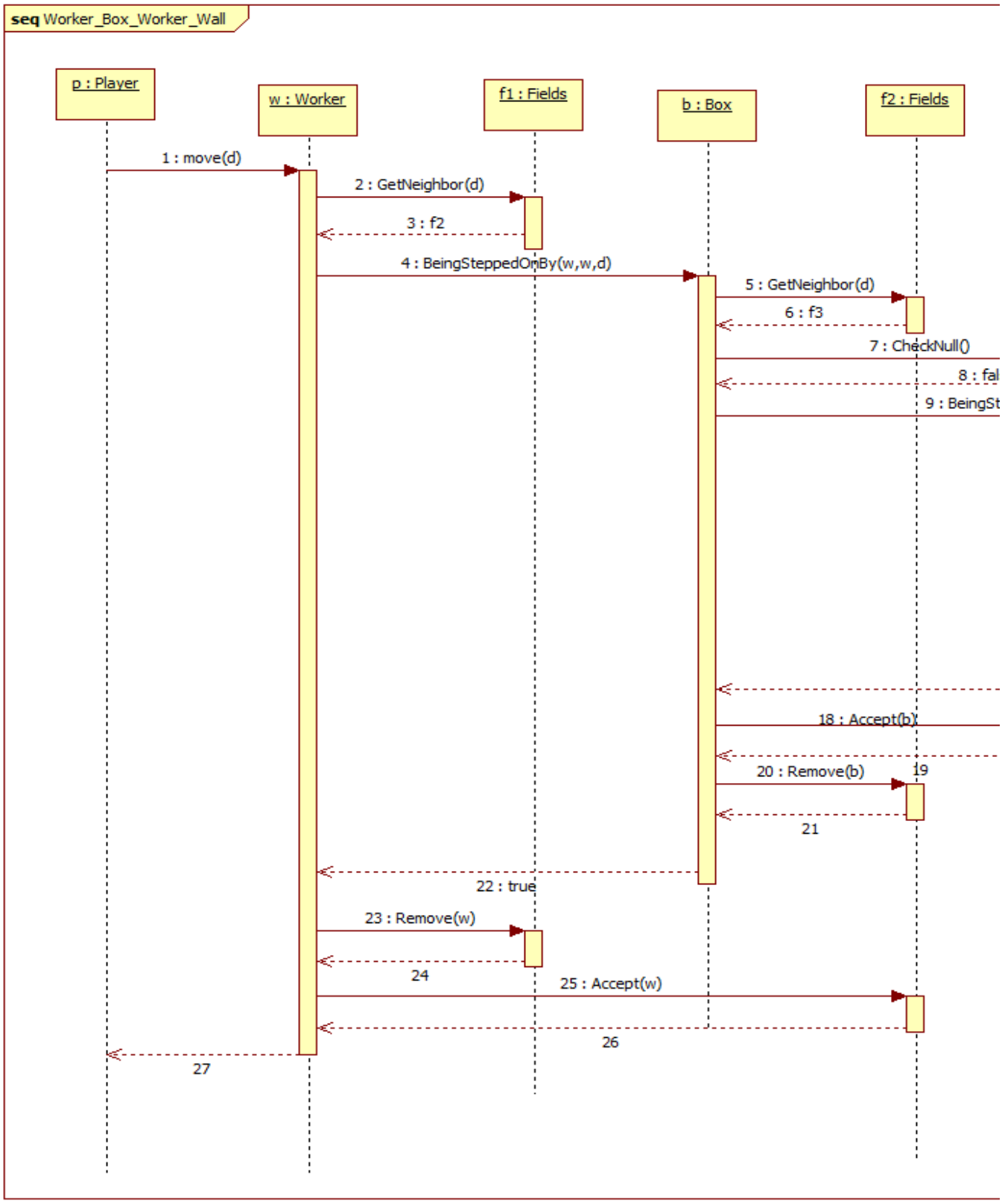
3.4.8 Worker_Wall

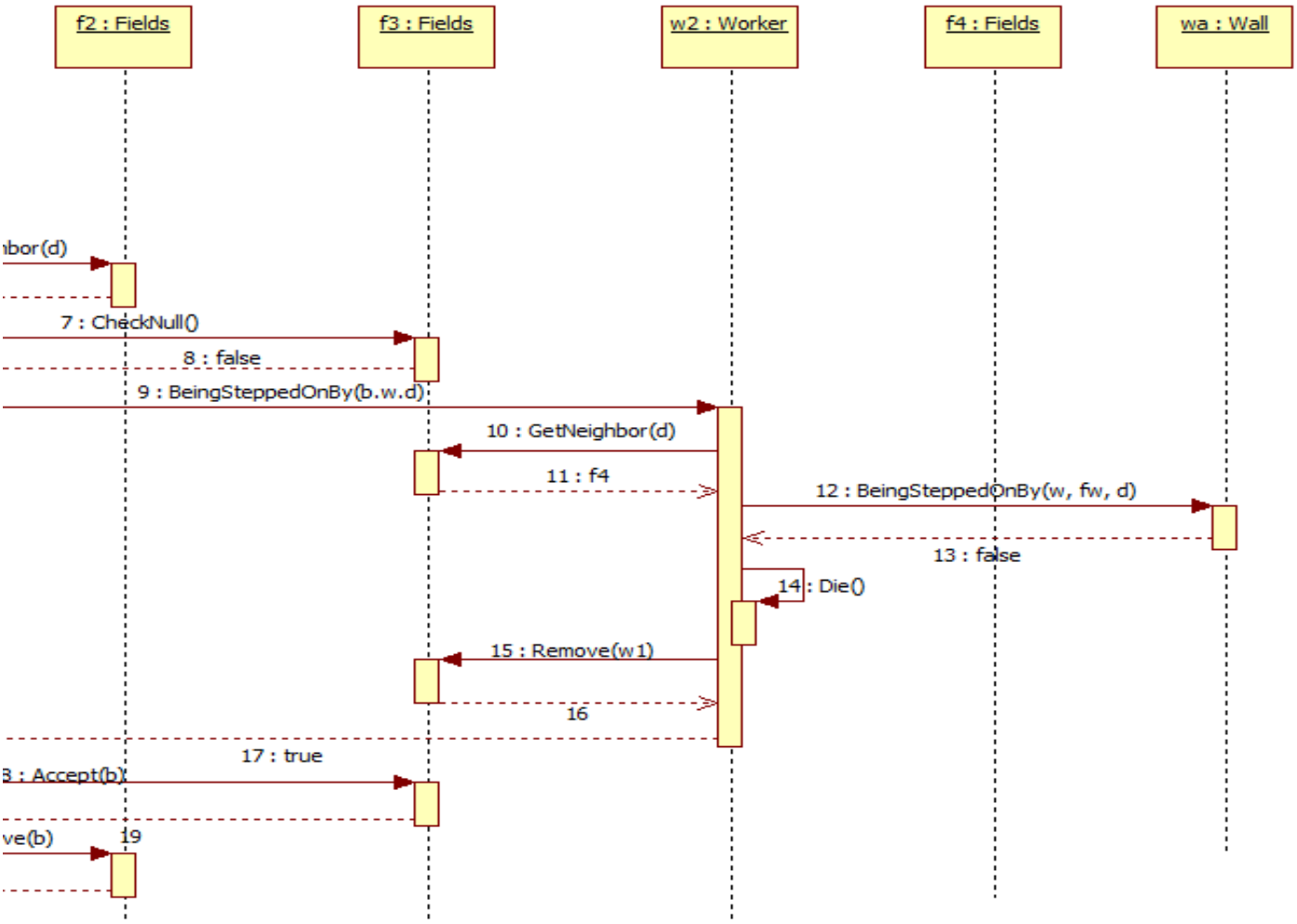
This Sequence diagram shows a worker try to move and there is a wall.



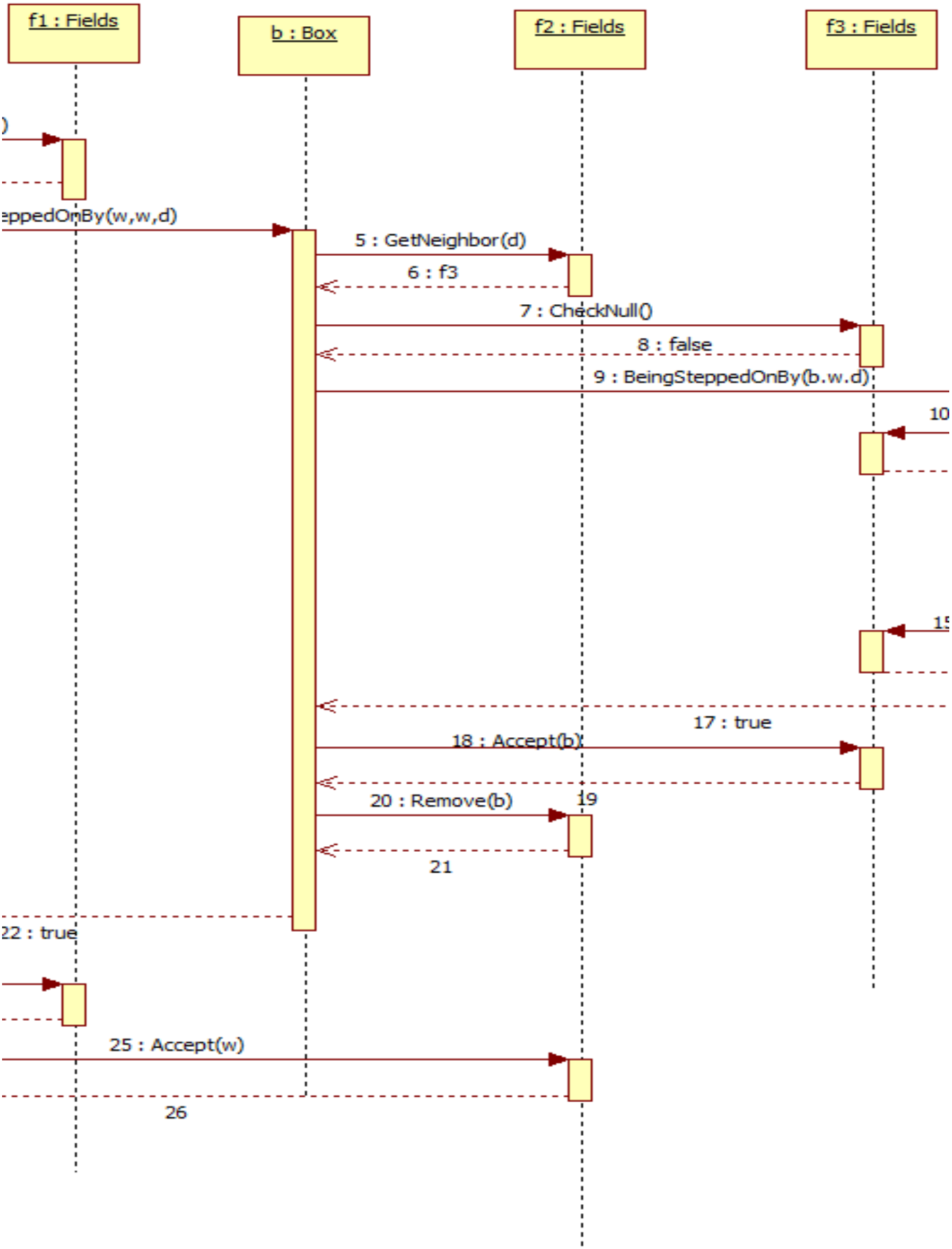
3.4.9 Worker_Box_Worker2_Wall

When a worker is pushed onto a wall by a box he dies and other entities are moved to their corresponding positions.



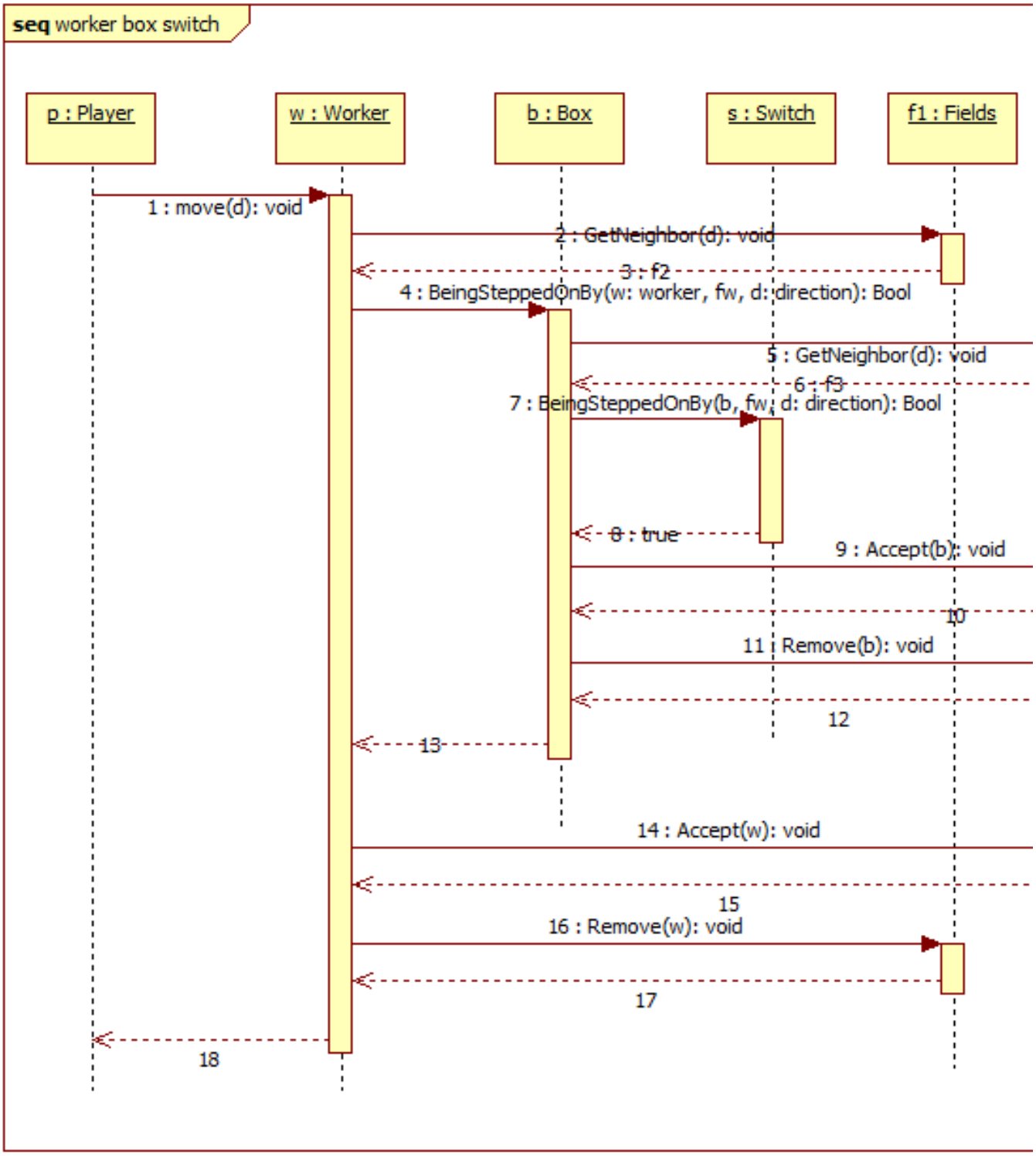


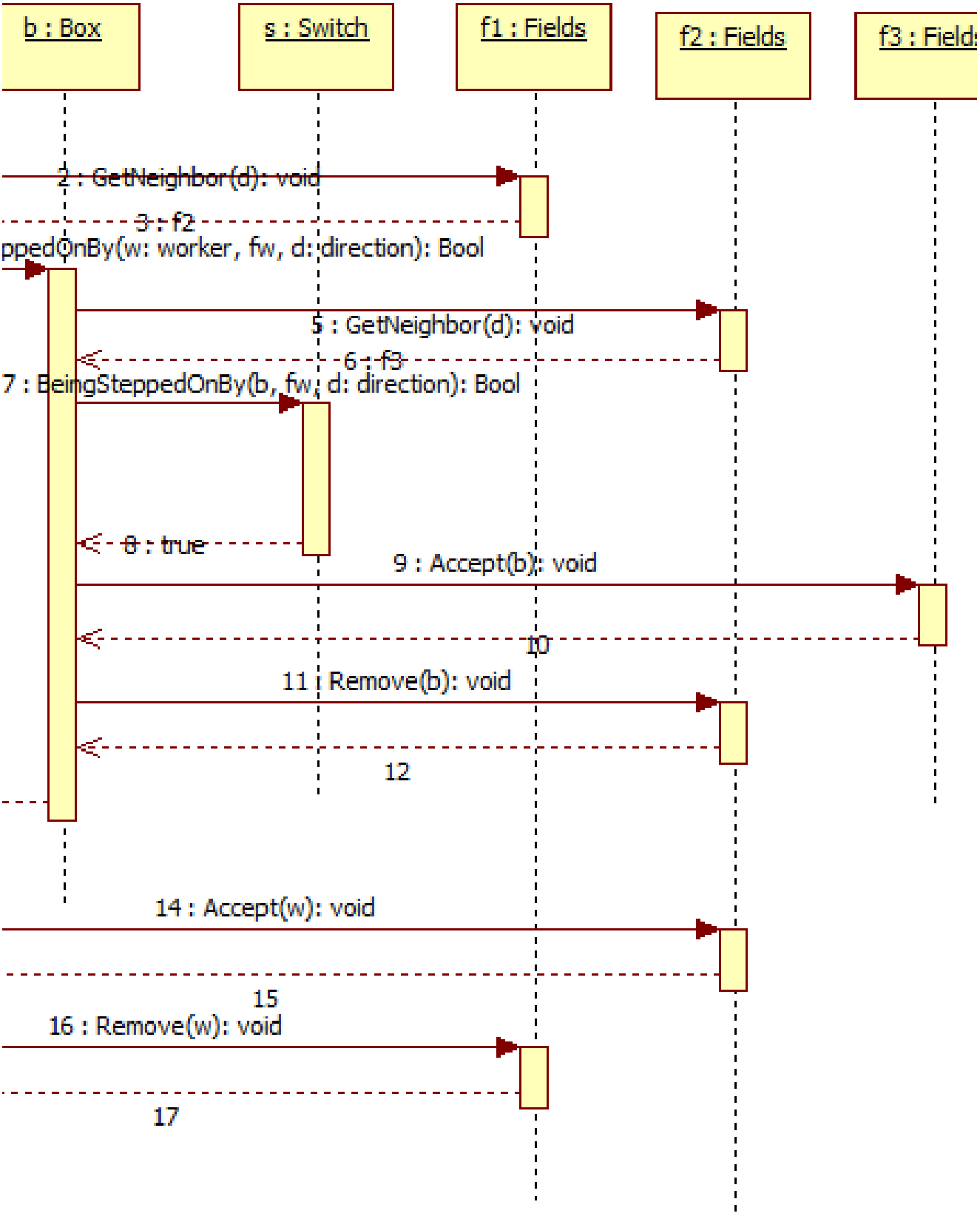
In the diagram getter and setters between associated objects have not been shown in some cases but .Also the function CheckNull() function of the Fienlds class which check wheather the Things corresponding are both Null pointest is also a just a simple getter call and a if check therefore in some cases where this is redundant it is skipped in the cases where this is written indicates the main focused part of the project for this exact sequence diagram.When the function call is not made means that this case was covered in some other seq diagram as you may have observed the number of distinct interactions between things can be 12 because 2 of out things namely, Box and Player are „ moving ” parts of the design they can each interact with 6 types of things .

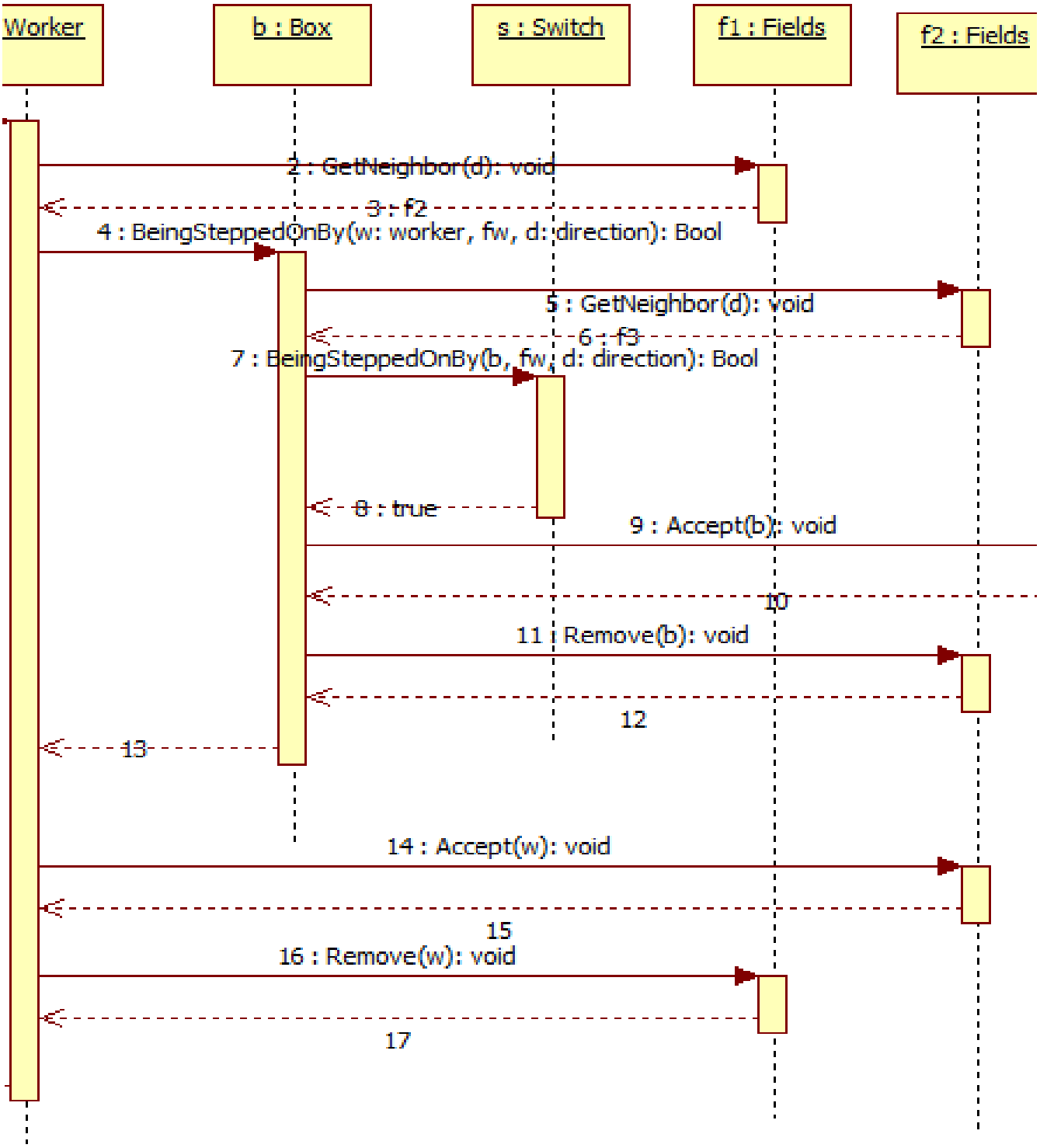

f1 : Fields
b : Box
f2 : Fields
f3 : Fields
eppedOnBy(w,w,d)
5 : GetNeighbor(d)
6 : f3
7 : CheckNull()
8 : false
9 : BeingSteppedOnBy(b.w.d)
10
17 : true
18 : Accept(b)
20 : Remove(b)
19
21
22 : true
25 : Accept(w)
26

3.4.10 Worker_Box_Switch

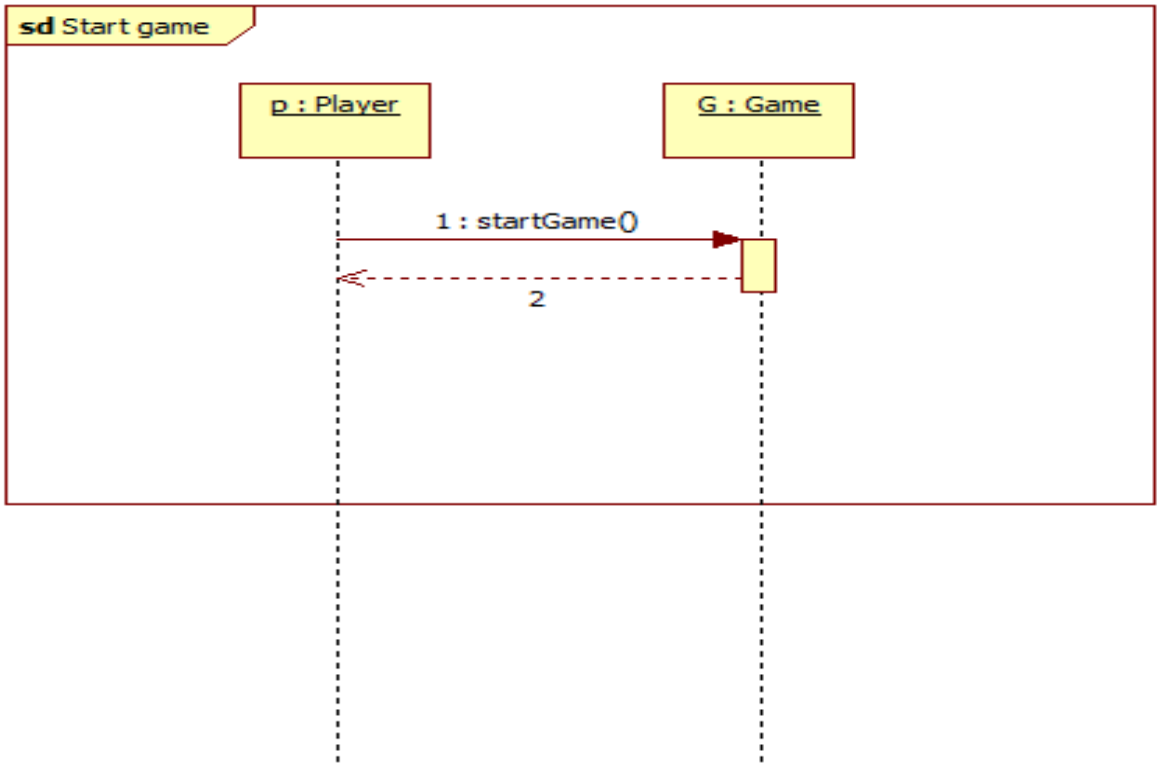
When a worker pushes a box or boxes onto the switch the inner state of the corresponding switch and the asoociated hole will be changed i havent included the inner state change of these objects because they are handled by setters and getters .




b : Box
s : Switch
f1 : Fields
f2 : Fields
f3 : Fields
2 : GetNeighbor(d): void
3 : f2
ppedOnBy(w: worker, fw, d: direction): Bool
5 : GetNeighbor(d): void
6 : f3
7 : BeingSteppedOnBy(b, fw, d: direction): Bool
8 : true
9 : Accept(b): void
10
11 : Remove(b): void
12
14 : Accept(w): void
15
16 : Remove(w): void
17


Worker
b : Box
s : Switch
f1 : Fields
f2 : Fields
2 : GetNeighbor(d): void
3 : f2
4 : BeingSteppedOnBy(w: worker, fw, d: direction): Bool
5 : GetNeighbor(d): void
6 : f3
7 : BeingSteppedOnBy(b, fw, d: direction): Bool
8 : true
9 : Accept(b): void
10
11 : Remove(b): void
12
13
14 : Accept(w): void
15
16 : Remove(w): void
17

3.4.11 Start game



3.5 State-charts

[Optional point. State charts can be attached ONLY to those classes which provide stateful behavioral features. State chart containing only ONE state is not allowed.]

3.6 Protocol

Start (date & time)	Duration (hours)	Performer(s) name	Activity description
[21.02.2018. 13:00]	*[4 hours]*	*All team's members*	[Meeting]
[23.02.2018. 14:00]	*[4 hours]*	*All team's members*	[Meeting]
[25.02.2018. 11:00]	*[12 hours]*	*All team's members*	[Meeting]

3. Analysis model – version 1

Object catalog

Fields, Map, Thing, Game, Player, Worker, DesignatedSquare, Wall, Hole, Switch, Box.

3.1.1 Fields

Fields represent the squares in the task description, the other interactive elements of the game will be associated with these and in some way placed on them. Their responsibility is to be able to get information about neighboring fields and either acccepting or removing objects from themselves. Every Thing element must have one Field and uses it in almost every case of the game to fulfill its requirements. It contains at most two Thing objects which can be accessed through the Getters and Setters.

3.1.2 Map

Map is a collection of Fields and is responsible for placing things and positioning them in some fields according to the specifications.

3.1.3 Player

The player represents the external actor of the game. Every player selects a Worker to play with, and that what makes the connection between the outside world and the game. It's connected to the DesignatedSquare which determines the amount of points a player gets in the game.

3.1.4 Worker

It represents the actual Worker who's objective is to move the boxes. It's controlled by one Player who's playing the game. It has the responsibility of initiating movements which might affect other objects.

3.1.5 DesignatedSquare

Represents the positions which the actor has to push boxes on. It's also responsible for checking the state of the object it shares the field with. It determines how players receive or lose points.

3.1.6 Wall

Repserents pillars or walls in the description. Cannot be moved or dislocated in any way once placed in the game's beginning.

3.1.7 Switch

A hole can be associated with a switch. If a hole has a switch then it's status can be turned on/off by moving a Box on a Switch. When a switch is turned on, the hole associated to it acts as a normal Field, and when it's off it acts as a Hole.

3.1.8 Box

Represents a Box on the field. When the game starts the Map generates Boxes and Designated Squares for them. The main objective of the game is to move these Boxes to the

Designated Squares, which will add points to the Players. These boxes can only be moved by Workers.

3.1.9 Hole

A Hole is associated with a Switch, which is off by default. When a Worker or a Box are moved into the hole, they die and get removed from the game. When the switch is on, a Worker or a Box can step on it as a normal Field.

3.1.10 Game

It handles the overall process of the game start and end. Also it manages the actual Players that are controlling the Workers.

Class description

One of the main entities is Thing which is an abstract class. A Thing is anything that can be placed on the squares of the floor.

A Thing can be be stepped on by a woker or a a box, this behaviour is haddeled by HitByBox method and HitByWorker which are purely virtual methods and will be implemented by its children with different implementation in each.

Thing is generalization of the following entities: Worker, Box, Hole, Switch, Walls(we decided that pillars and walls are the same) and DesignatedPlace.

We deciced that the best way to resolve the problem is by considering everthing an object that can be placed on squares(fields).

Fields in other terms square is another main entity: it can accept a Thing on it as well as removing it, it can also get its neighbors and returns a pointer referencing the Thing on it. On a field we can have either 0, 1 or 2 Things on it, the class implements a method checkNull, that return a bool and tells whether there is a thing on the field or not.

3.2.1 Box

- **Responsibility**

  A box can move to a given direction. A box dies when it gets into a hole. A Player receives a point when he moves the box to a Designated Square. It also remenberes a reference to the object(Thing) which pushed it.

- **Superclasses**

  -*Thing*

- **Interfaces**

  -

- **Attributes**

  Public previous: Thing : This attribute is a reference to the Thing which may call this object by the **HitByBox** or **HitByWorker**, it is by default Null and will only change while the **HitByBox** and **HitByWorker** is active for the current worker(this). It may be

asked by the GetPusher(t: Thing) of the **DesignatedSquare** when adding points to the player**.**

- **Methods**

  - **Public void die() :** the box gets destroyed/removed. It can only be called by a hole when the box falls onto the hole.

  - **Public bool HitByWorker (w:Worker, d:Direction) :** The current object(this Box) is asked to relocate by the other object of type Worker passed as a paramater (w). The current box will ask the object(neigbour) in the direction of the push (d) with the same function but it will pass itself as the box parameter of the HitBtBox(this,d) if successful (the asked object will return true) then will relocate, change its **(previous :Thing)** parameter to the passed object (w) and fianlly return (true) if not then will remain in the same field and return (false).

  - **Public bool HitByBox (b:Box, d:Direction):** The current object(this box) is asked to relocate by the other object of type Box passed as a paramater (b). The current box will ask the object(neigbour) in the direction of the push (d) with the same function but it will pass itslef as the box parameter of the HitBtBox(this,d) if succesful (the asked object will return true) then will relocate, change its **previous :Thing** parameter to the passed object (w) and fianlly return (true) if not then will reamin in the same field and return (false).

3.2.2 DesignatedSquare

- **Responsibility:**

Adding points to the player whose worker delivered a box to a designated place. Substracts points to the player whose worker moves a box that reached a designated place. It is also capable of determining which player pushed first in a specific push given a direction.

- **Superclasses:**

Thing

- **Attributes:**

- **Methods:**

  - **public void addPoints(p:Player):** when a worker gets a box to a designated place, this method will be executed to add points to its corresponding player, the player passed by parameters.

  - **Public void removePoint(p:Player):** when a worker moves a box from a designated place, its corresponding player will lose points, and this method handles the subtraction of points.

- **Public Thing GetPusher(t: Thing):** This method is used to get a reference to the object(thing) which made the last push to the asked object. When a box moves onto a DesignatedSquare then this method will ask the box for its's Box's **previous** pusher which will return another Thing type reference(can be worker or box does not matter) then we will again do the same prosedure as long as we dont get a Null. The worker by definition of the requirements is the object of type thing which started the push which means that the first object that returns Null for its **previous** is the first worker who started the push. This function will be used whenever determining which player we need to add to or remove points from is requested.

3.2.3 Fields

- **Responsibility**

-Stores Things that can appear in the Warehouse.
-Fields knows its neighbours.
-Represents the Squares in the floor of the warehouse.

- **Superclasses**
  -

- **Interfaces**
  -

- **Attributes**
  - **Neighbors[Direction]:** stores the neighbouring fields in the possible directions.
  - **-thing:Thing[0..2]:** Things currently on the Field; it can be empty, 1 object, or 2 objects on it.

- **Methods**
  - **Public void Accept(t:Thing):** accept the t:Thing on the current field using its getters and setters.
  - **Public void remove(t:Thing):** remove the t:Thing from the current field field using its getters and setters.
  - **Public Fields getNeighbour(f1:Fields, d:Direction):** returns the Fields according to the given direction.
  - **Public void setField(d:Direction, f:Fields):** Given a field sets a field as a neighboring field.
  - **Public bool checkNull():** tells if there is something on the Field or not, if the Field is empty returnes (true), and (false) otherwise.
  - **Public Thing GetThing(i: int):** returnes the Thing from the list of things on the field int **i** will show the
  - **Public SetThing(i: int, t: Thing):** sets the thing in the index (i) which can not be more than 2 with the passed Thing.

3.2.4 Game

- **Responsibility**

Manages the game starting and ending, also adds the Players which are the major actors in the game.

- **Superclasses**

-

- **Interfaces**

-

- **Attributes**

-

- **Methods**
  - **Public void addPlayer(p:Player):** adds a Player to the game.
  - **Public void startGame():** starts the game.
  - **Public void endGame():** ends the game.

3.2.5 Hole

- **Responsibility**

If a worker or a box step on a the hole, so it dies.

- **Superclasses**

Thing

- **Interfaces**

-

- **Attributes**
  - **public State : boolean:** it gives the state of the hole, meaning if the switch corresponding to this Hole is on, then the hole acts as a normal Field and returns (false) and if the switch is turned off then it acts as a Hole and returns (true).

- **Methods**
  - **Public GetSwitch:** Switch: Returns the Switch corresponding to this Hole.

3.2.6 Map

- **Responsibility**

-Stores the Fields of the game which are the blocks making the grid of the Map.
-Adds all the Things to the game: Box, Worker, Hole, Switch, Wall, Designated Place.

- **Superclasses**

Thing

- **Interfaces**
  -
- **Attributes**
  -
- **Methods**
  - **Public void addBox(b:Box):** adds a Box to the grid.
  - **Public void addSwitch(s:Switch):** adds a Switch to the grid.
  - **Public void addWorker(w:worker):** adds a Worker to the grid.
  - **Public void addHole(h:Hole):** adds a Hole to the grid.
  - **Public void addWall(w:wall):** adds a Wall to the grid.
  - **Public void addDesignatedSquare(d:DesignatedSquare):** adds a DesignatedSquare to the grid.

3.2.7 Thing

- **Responsibility**:

Represents a general thing (e.g. Wall, Hole, Switch, Box, Worker, Designated Square) which can be on a Field.

- **Superclasses**
  -
- **Interfaces**
  -
- **Attributes**
  -Field: Fields → The Field the thing is on.
- **Methods**
  - **HitByBox(b: Box, d: Direction): boolean:**
    - Indicates a push which is being made by Box **b** in direction **d** The result is determined in each case differently as per each subclass of thing. Look at a specific subclass for detailed explanation.

  -HitByWorker(w: Worker, d: Direction): boolean:
    - Indicates a push which is being made by Worker **w** in direction **d** The result is determined in each case differently as per each subclass of thing. Look at a specific subclass for detailed explanation.

  - GetField(): Fields:
    - Return the field of the Thing . Which the object(Thing) is standing on , associated with.

  - SetField(): Fields:

- Set the field of the Thing .Which the object(Thing) is standing on , associated with.

3.2.8 Switch

- **Responsibility**

Change the state of an associated Hole, in order to act as a normal Field or as a Hole.

- **Superclasses**

-*Thing*

- **Interfaces**

-

- **Attributes**
  - **Public State: boolean:** returns if a switch is turned ON or OFF.

- **Methods**

-**Public GetHole: Hole:** Returns the Hole corresponding to the Switch.
-**Public SetHoleState(s: boolean):** Sets the state of corresponding Hole by setting it ON or OFF, which is passed in the argument as true/false.

3.2.9 Wall:

- **Responsibility**

-A Wall is an obstacle on a Field, it can be a Pillar or a Wall on the grid.
-It stops the movement of Things, you cannot step on it.

- **Superclasses**

-*Thing*

- **Interfaces**
  -

- **Attributes**
  -

- **Methods**
  - **Public bool HitByWorker (w:Worker, d:Direction)** When a worker **(w)** tries to move on the field which has a Wall, this function will be called and this will always return false because of the Wall (according to the description) which is unable to relocate the original position appointed by the map and can never change in the middle of the game.

  - **Public bool HitByBox (b:Box, d:Direction):** When a box **(b)** tries to move on the field which has this wall this function will be called and this will always return false because of the Wall (according to the description) which is unable to relocate the

original position appointed by the map and can never change in the middle of the game.

3.2.10 Worker

- **Responsibility**

-Can move along the grid with a given direction.
-Can push a Box or sequence of Boxes into a given direction.
-Dies if it gets pushed to a Wall or to a Hole.
-Remenberes a reference to the object(Thing) which pushed it.

- **Superclasses**

-Thing

- **Interfaces**

-

- **Attributes**

Public previous : Thing: This attribute is a reference to the Thing which may call this object by the **HitByBox** which is by default Null and will change while the **HitByBox** is active for the current worker(this). It may be asked by the **GetPusher(t: Thing)** of the **DesignatedSquare** when adding points to the player.

- **Methods**
  - **Public void die() :** the worker gets destroyed thus the game will end for its corresponding player.
  - **Public void move(d:direction):** moves the worker along the given direction. The direction is specified by the Player in control of the worker.
  - **Public GetPlayer: Player:** Returns the Player associated with the Worker.
  - **Public bool HitByWorker(w1:worker, d: Direction):** The current worker is being pushed by another worker which will result in no change for this worker. Because when relocation does not happen the return value is always false indicating to the caller that it cannot reallocate here.

    Public bool HitByBox(b: Box, d: Direction): The current worker (this) is being pushed by the box passed in the parameters **(b)**. It will return a boolean value depending on the success of the relocation of worker if worker is able to move then it will return true if the worker is unable to relocate in the given (passed) direction then he will die and return false.

3.2.11 Direction

- **Responsibility**

- This enumeration lists the possible neighboring directions of Fields: Up, Down, left or right. and also the direction a Worker or a Box can take: Up, Down, Left or Right.

- **Superclasses**

-

- **-Interfaces**
  -

- **Attributes**
  - Right
  - Left
  - Up
  - Down

- **Methods**

  -

3.2.12 Player

- **Responsibility**

-The Player is the main actor in the game, he's the one who decides all the moves to be made.
-Every Player is controlling one Worker which will iniate a move.

- **Superclasses**

-

- **Interfaces**

-

- **Attributes**
  - **Public points: int:** This attributes counts the number of points the Player obtained -> By moving Boxes into their designated places.

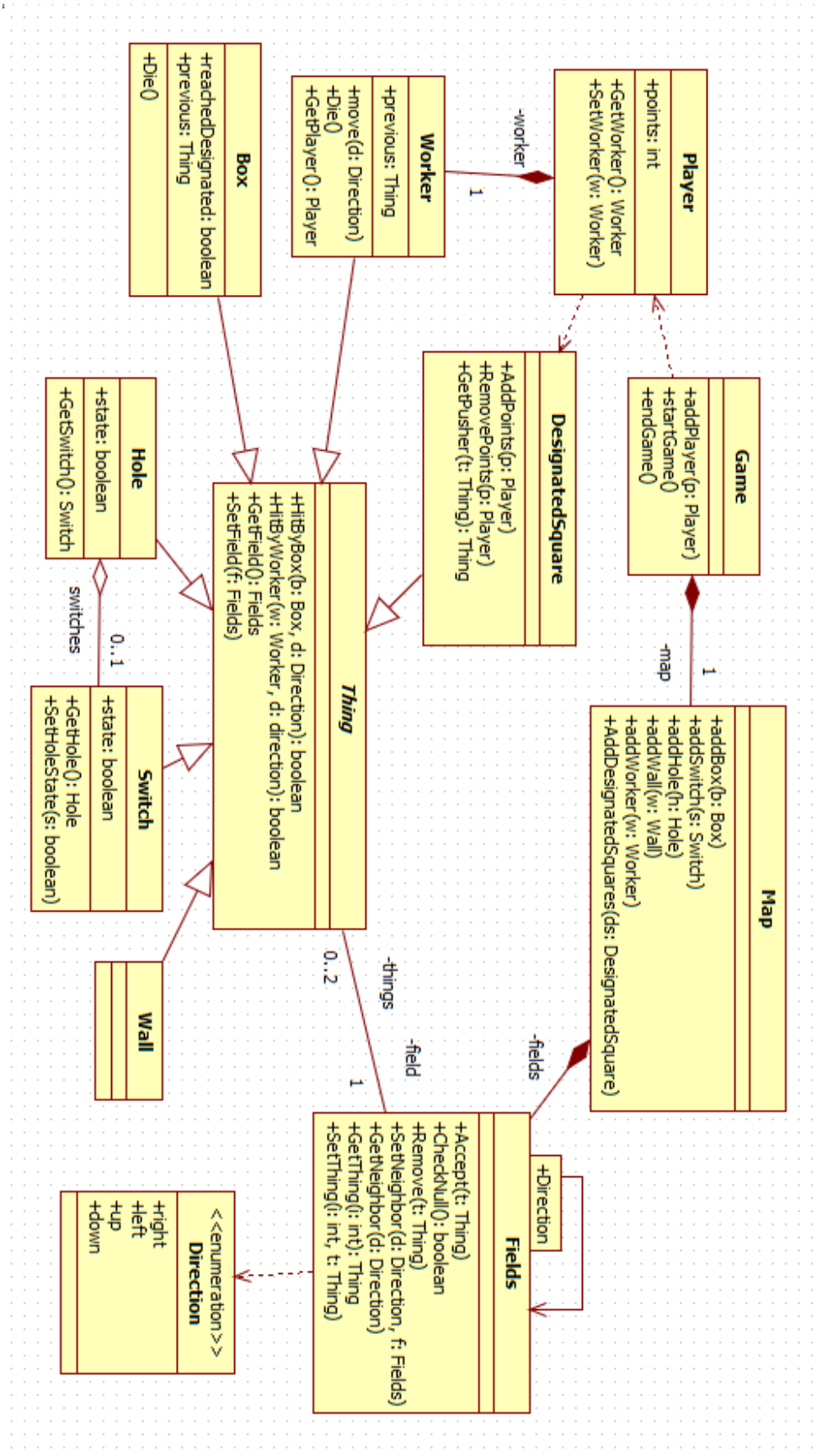
- **Methods**

-**Pulic GetWorker : Worker:** Returns the current Worker associated to the Player.

-**Public SetWorker(w: Worker):** Assigns the Worker (w) to the Player.

-**Public CheckPoint: int:** Returns the number of points that a Player has.

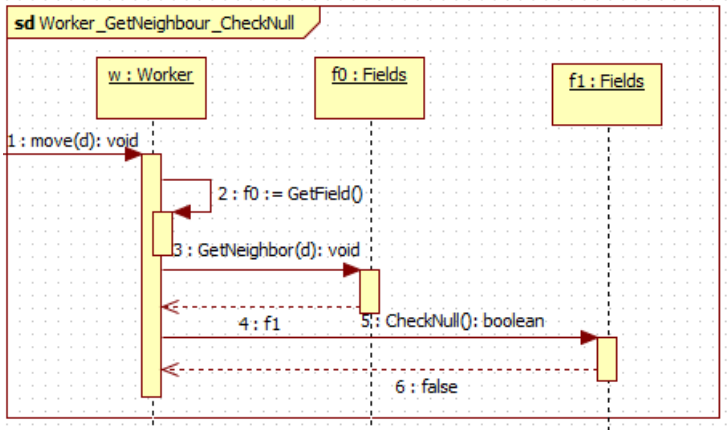
Static structure diagrams



Sequence diagrams

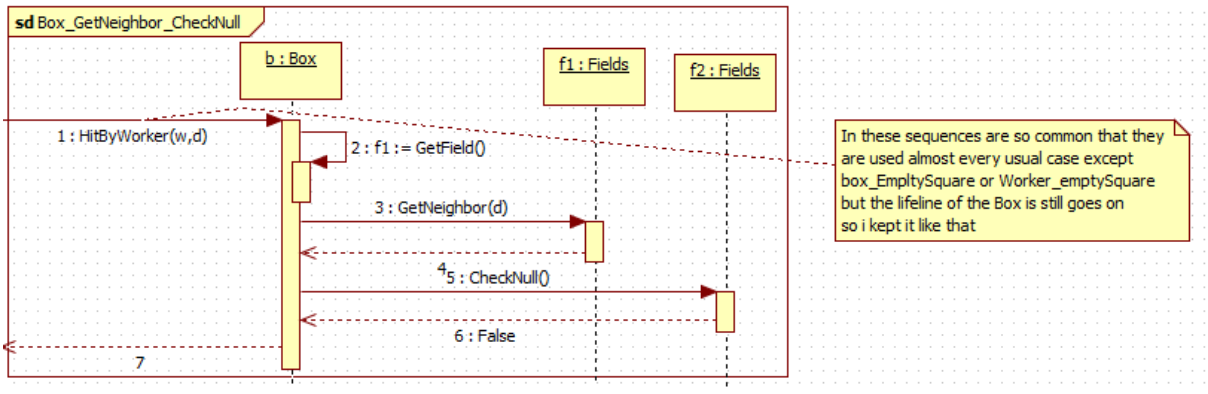
3.4.1 Worker GetNeighbor CheckNull reference

This Sequence Diagram shows the basic methods after a move function for the Worker, it's referenced in the diagrams later.



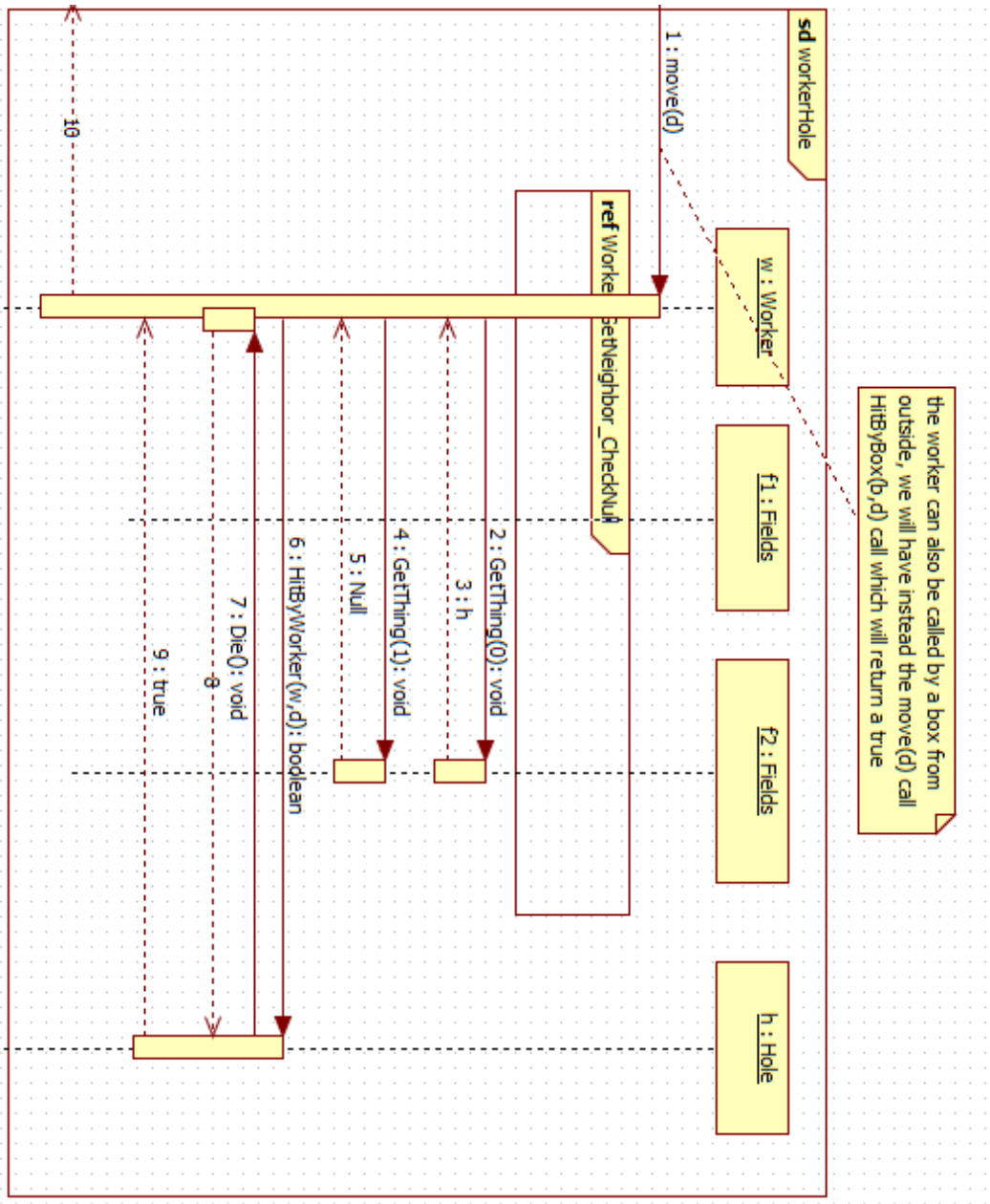
3.4.2 Box GetNeighbor CheckNull reference

This Sequence Diagram shows the basic methods after a move function for the Box, it's referenced in the diagrams later.



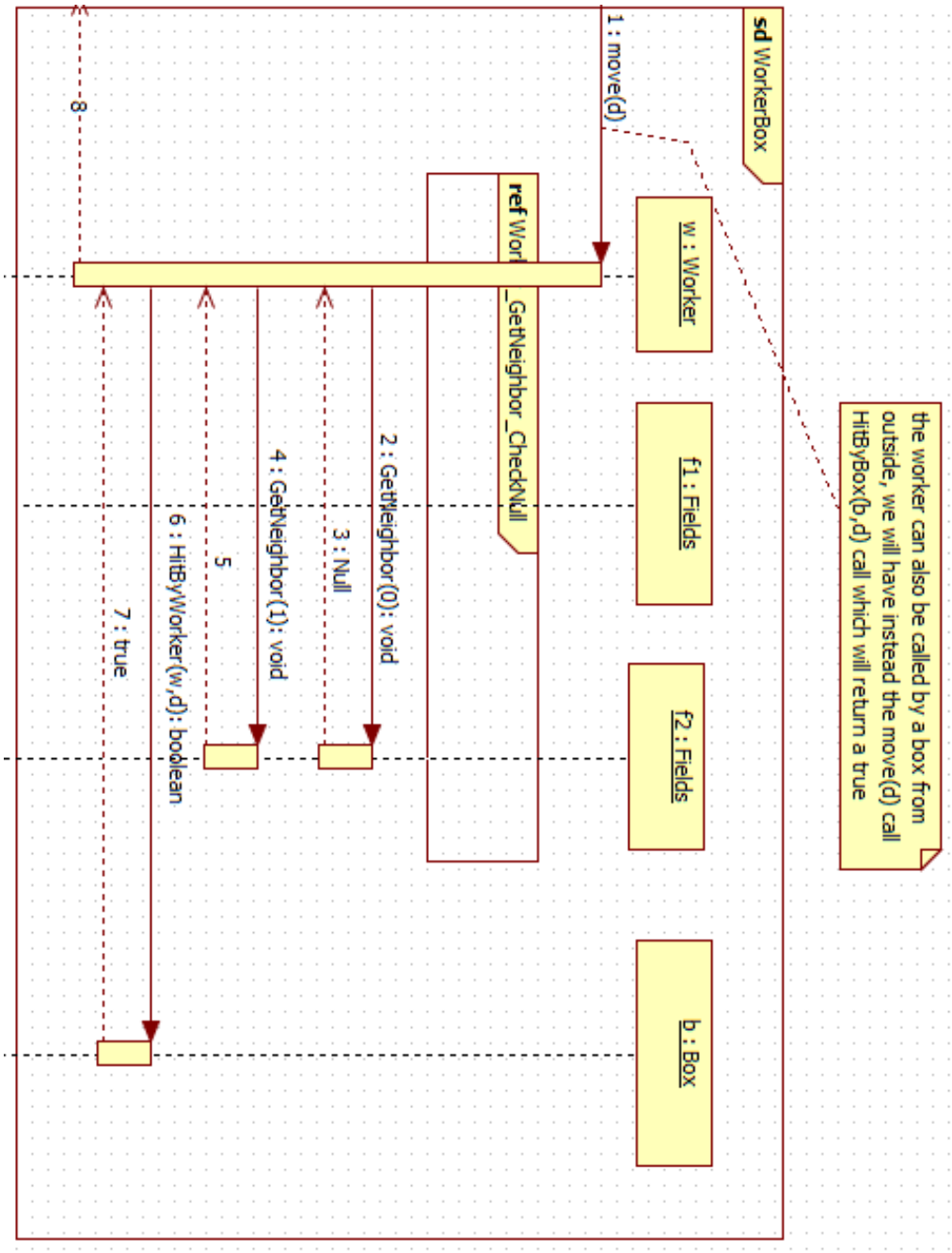
3.4.3 Worker Hole

This Sequence diagram shows when a worker moves to a hole.



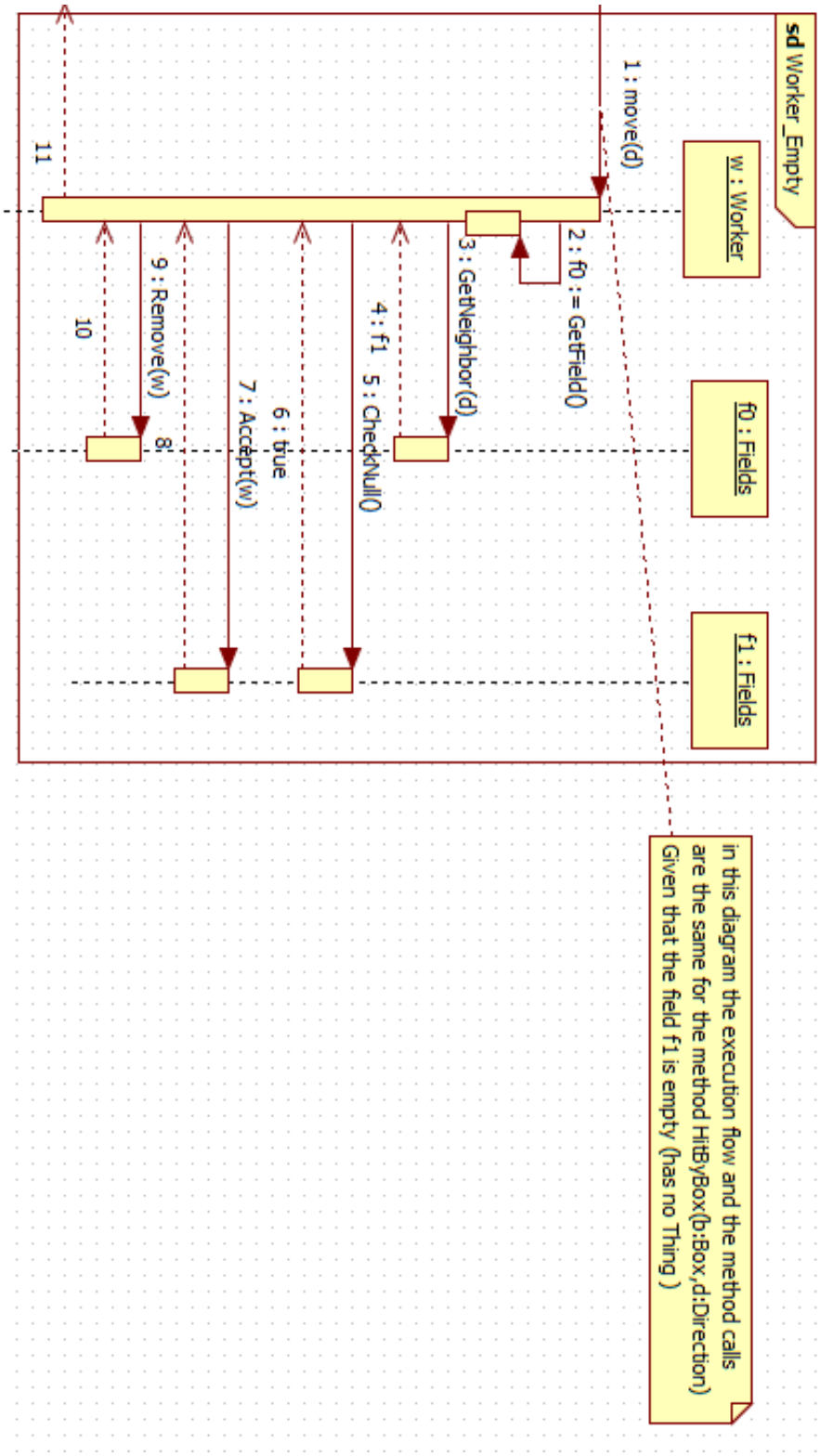
3.4.4 Worker Box

This Sequence diagram shows how a worker moves the box to an empty field.



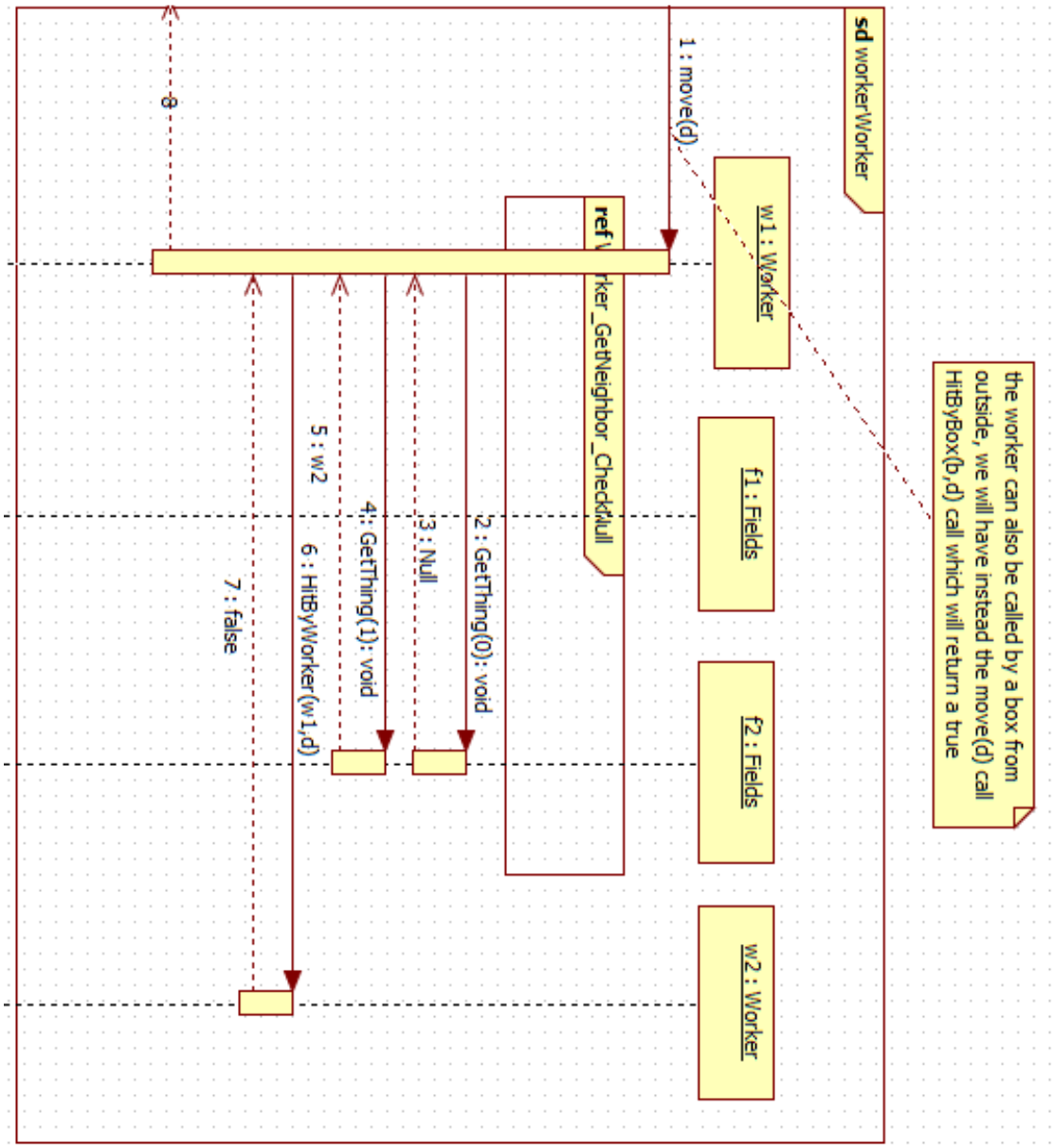
3.4.5 Worker Empty

This Sequence diagram shows how a worker moves into a give direction.



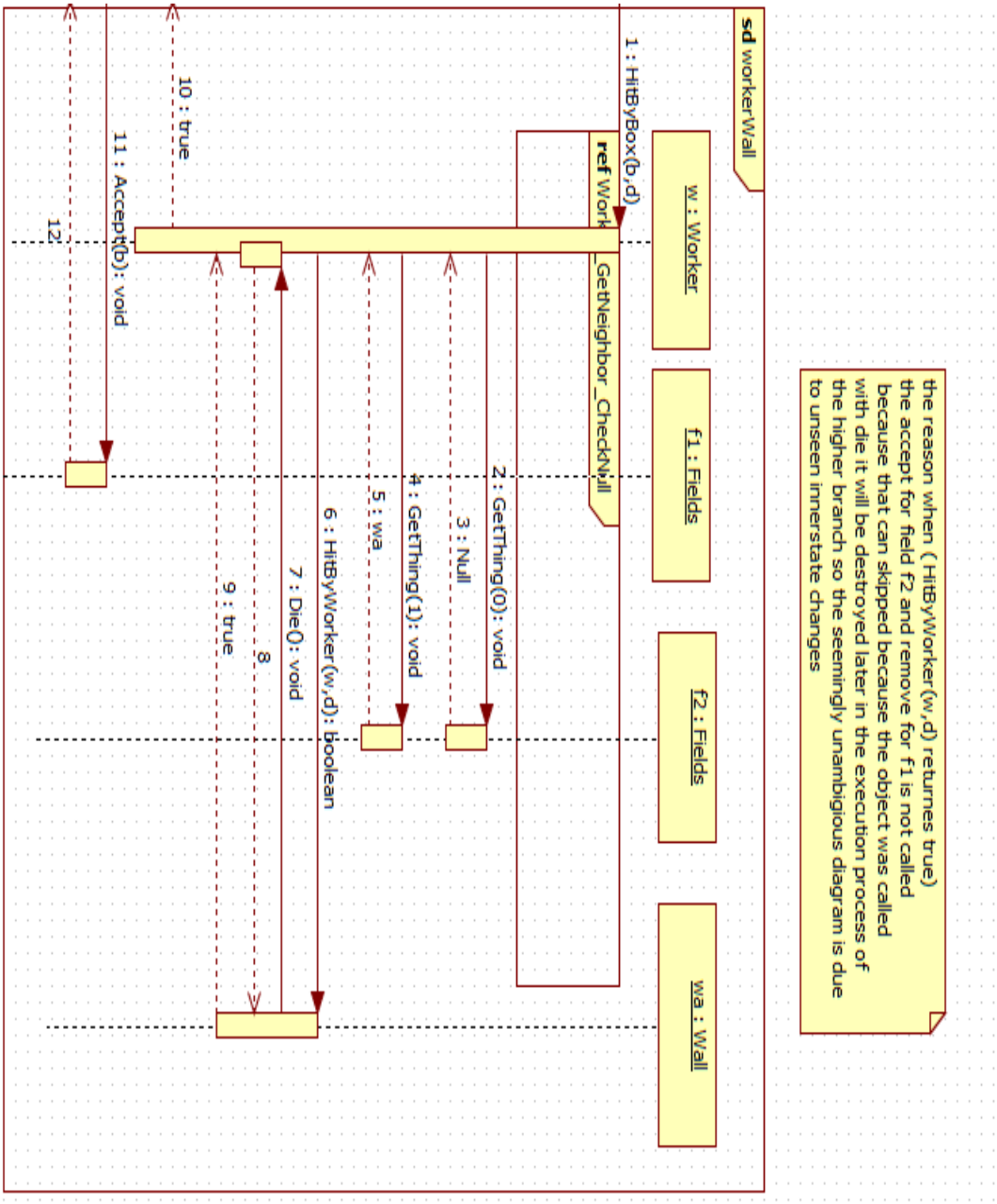
3.4.6 Worker Worker

This Sequence diagram shows how a worker pushs a worker.



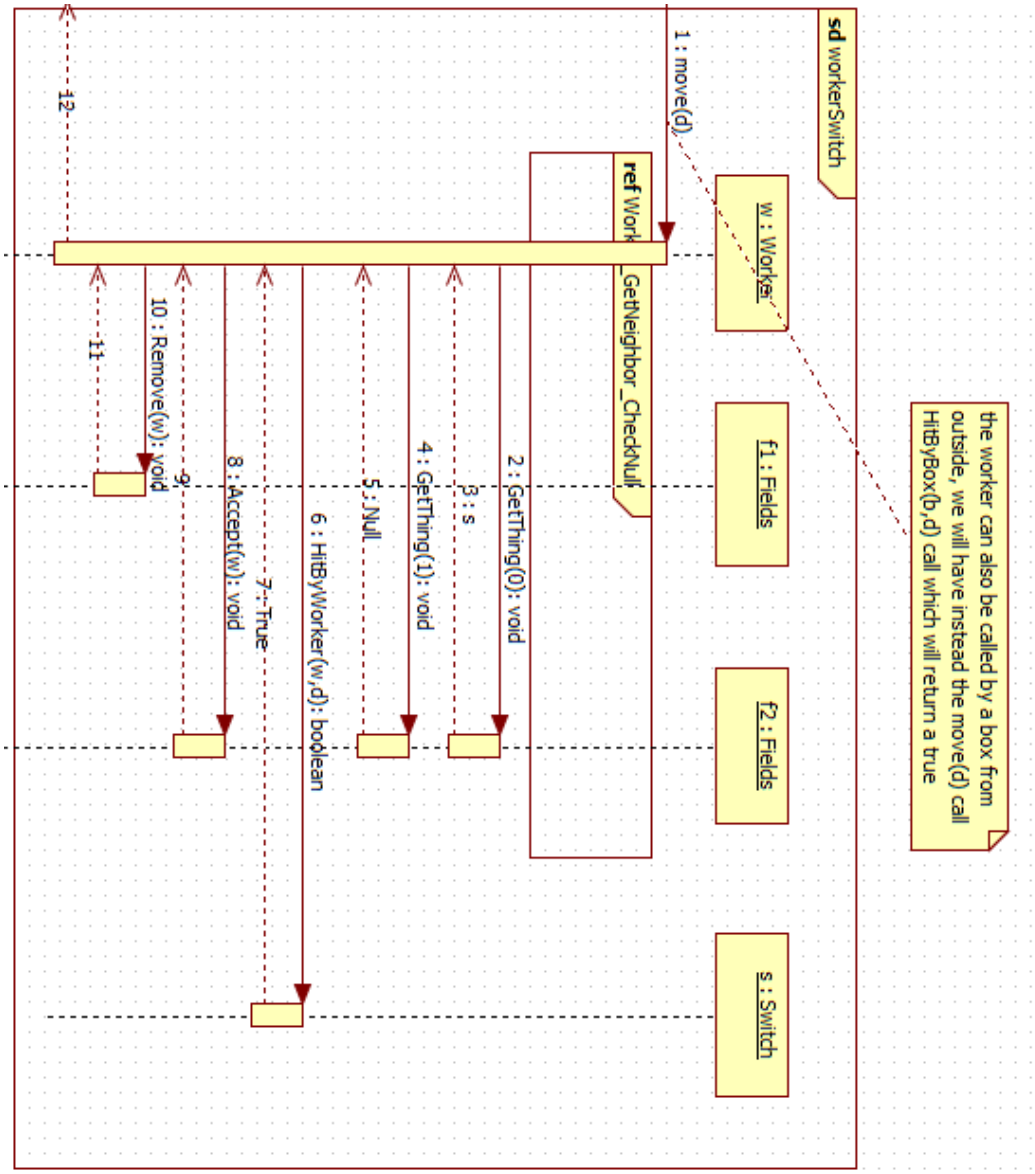
3.4.7 Worker Wall

This Sequence diagram shows a worker try to move and there is a wall.



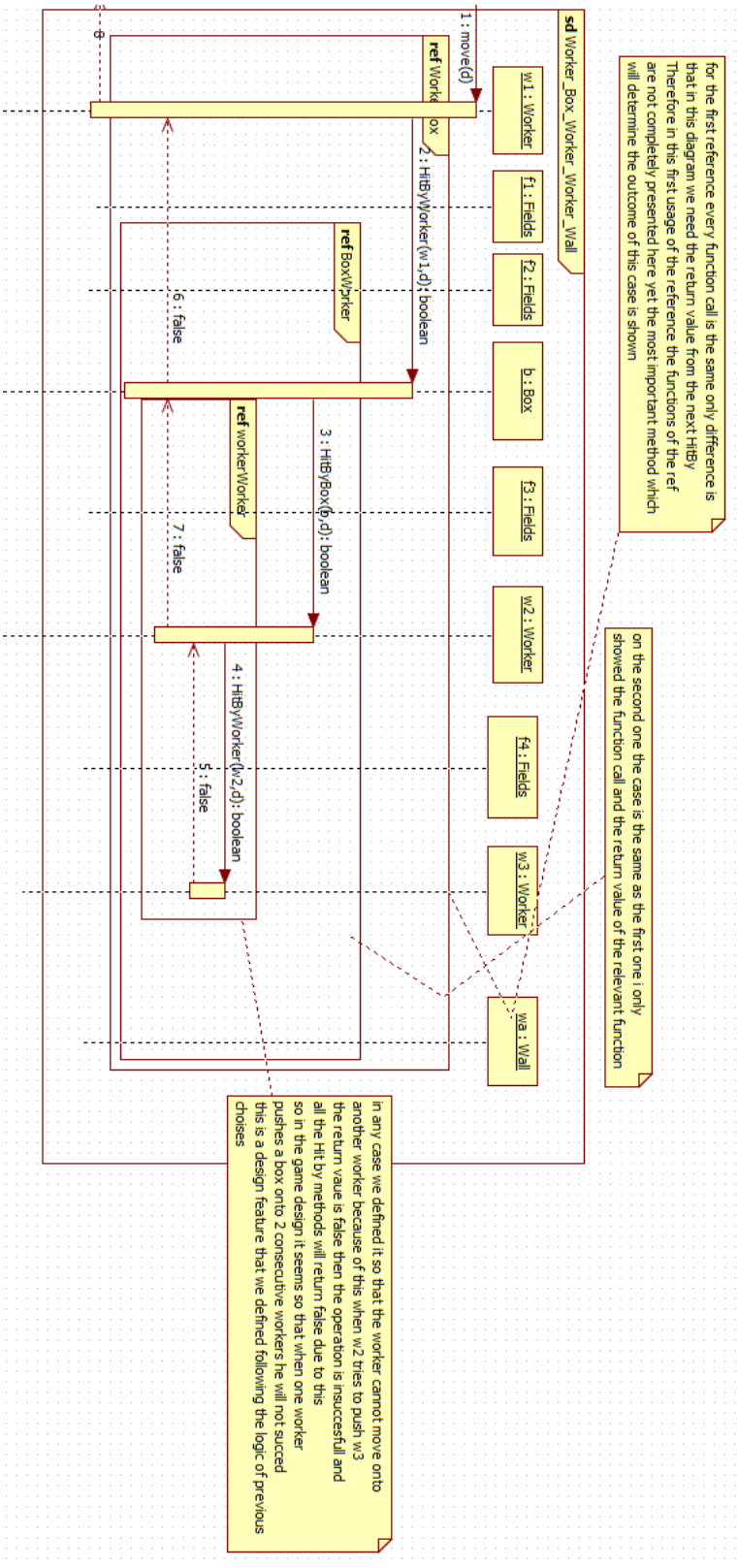
3.4.8 Worker Switch

This Sequence diagram shows a worker try to move and there is a Switch.



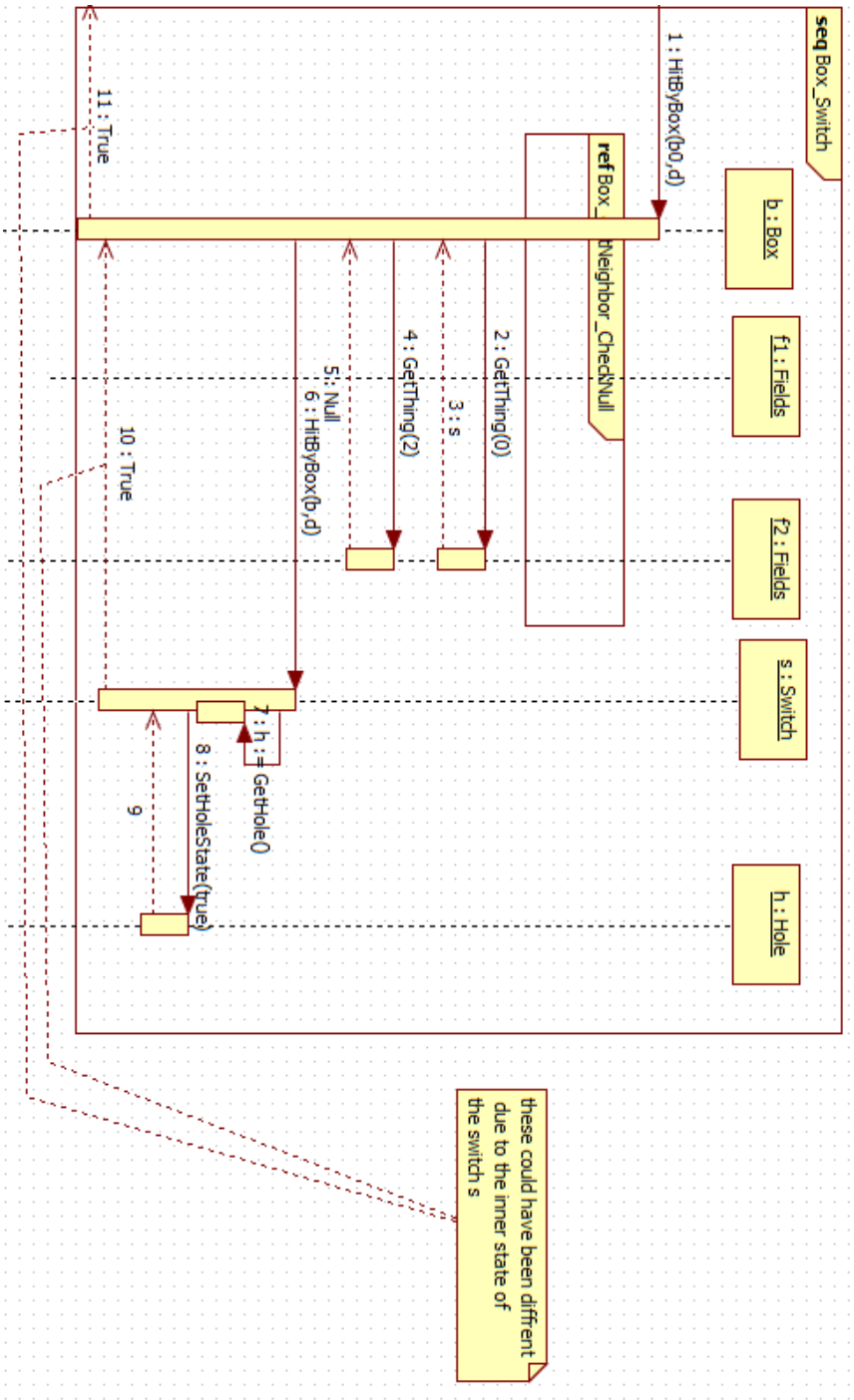
3.4.9 Worker Box Worker Worker Wall

When a worker is pushed onto a wall by a box he dies and other entities are moved to their corresponding positions.

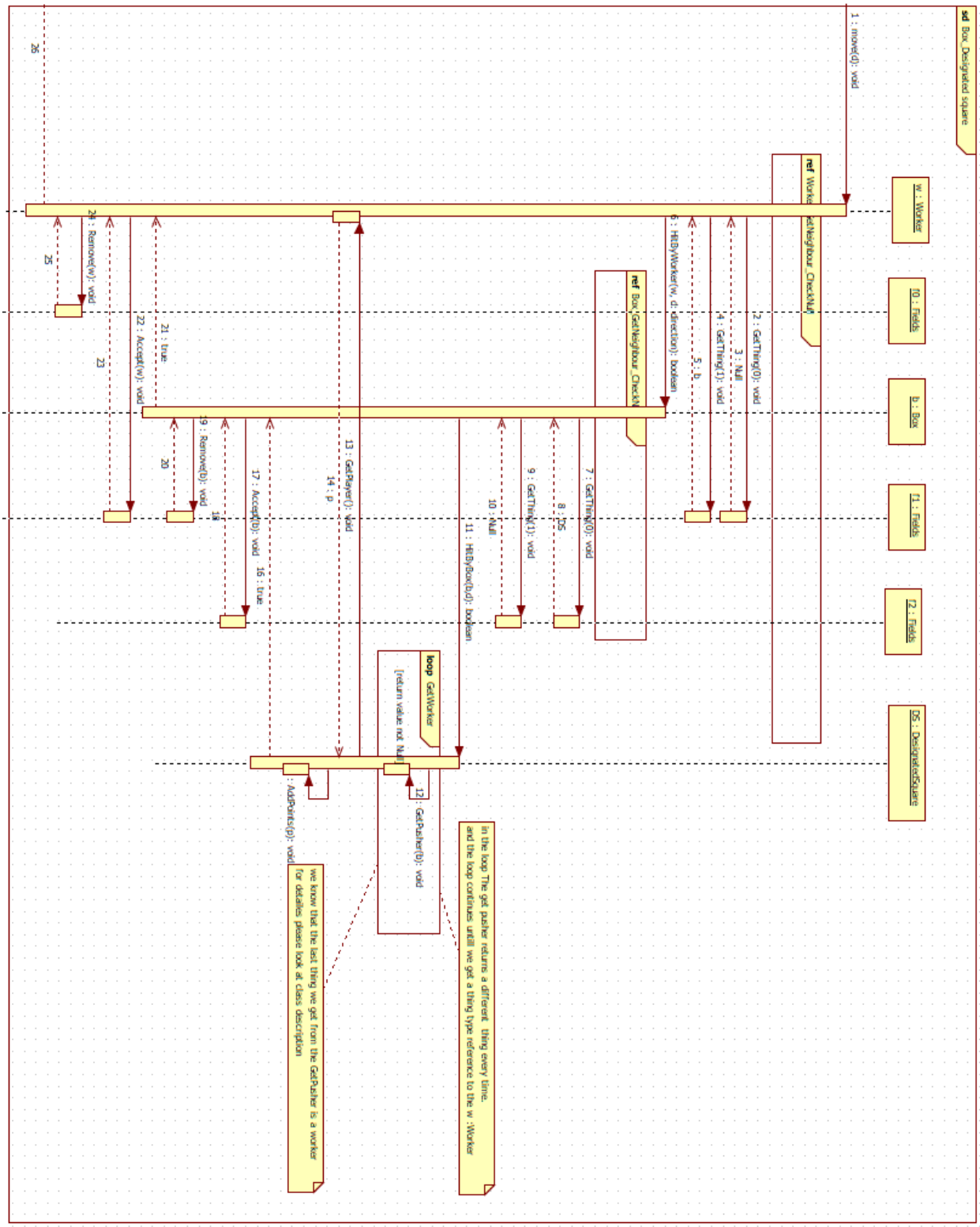


3.4.10 Box Switch

When a worker pushes a box or boxes onto the switch the inner state of the corresponding switch and the asoociated hole will be changed i havent included the inner state change of these objects because they are handled by setters and getters.



3.4.11 Box DesignatedSquare

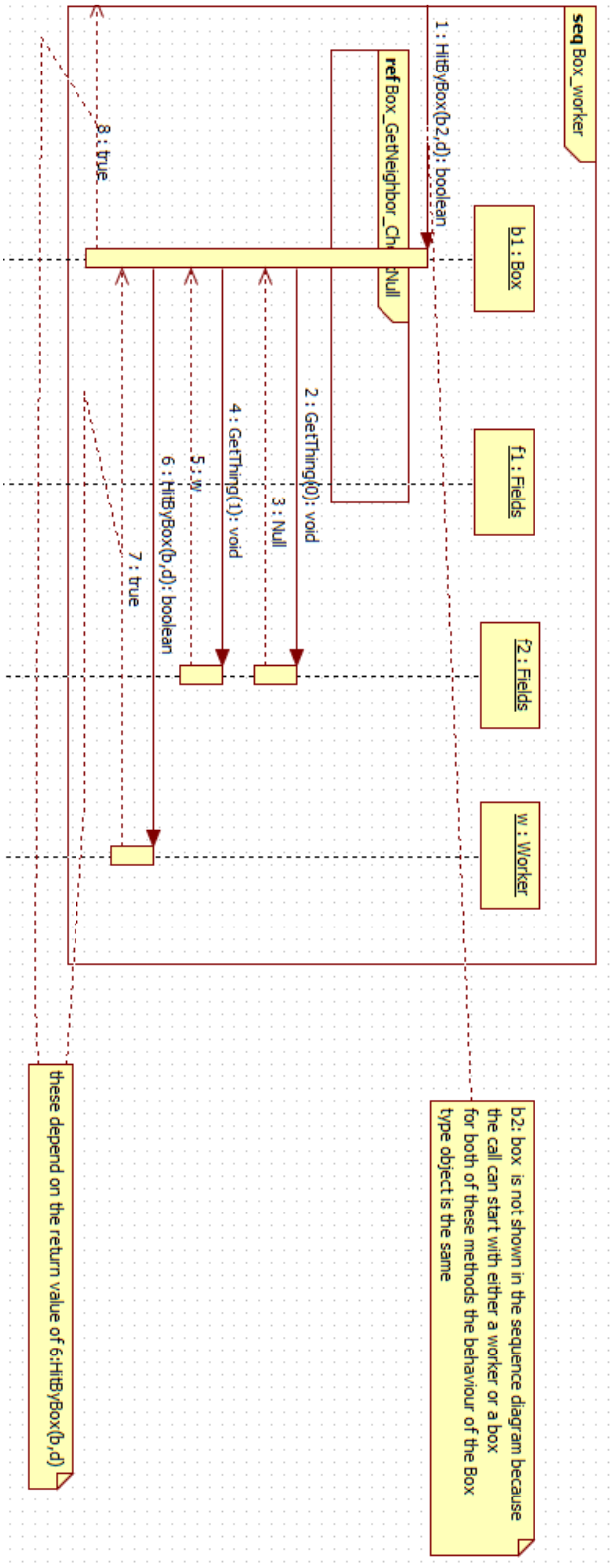


Box Designated Square:

Worker gets a move direction from the Player and checks the CheckNull condition assuming that it returns false we ask the already acquired neighbor field f1 for thing in index 0 and 1 then we get a Thing type reference to b from the perspective of the w:Worker the returned value is just a thing but we will call its HitByWorker(w,d) in which it will do almost the same and get a reference to DS with type thing it will call DS's HitByBox(b,d) and while in the Designated square first the passed box b will be asked its previous by GetPusher and it will return a Thing type reference to the object which pushed the box. This will be done as many times as necessary until we get a Null as return Value. When this happens we will stop the loop and ask the worker to return its Player(We know the last reference is of type Worker only a worker can start a push and the value of the previous only change in the HitByBox). After we Get the player, DS will add point to the player and return true to the function call 11.
After this every object that interacted will move accordingly first the box will relocate next to the DS on the dame field f2. Then worker will relocate and after that return.

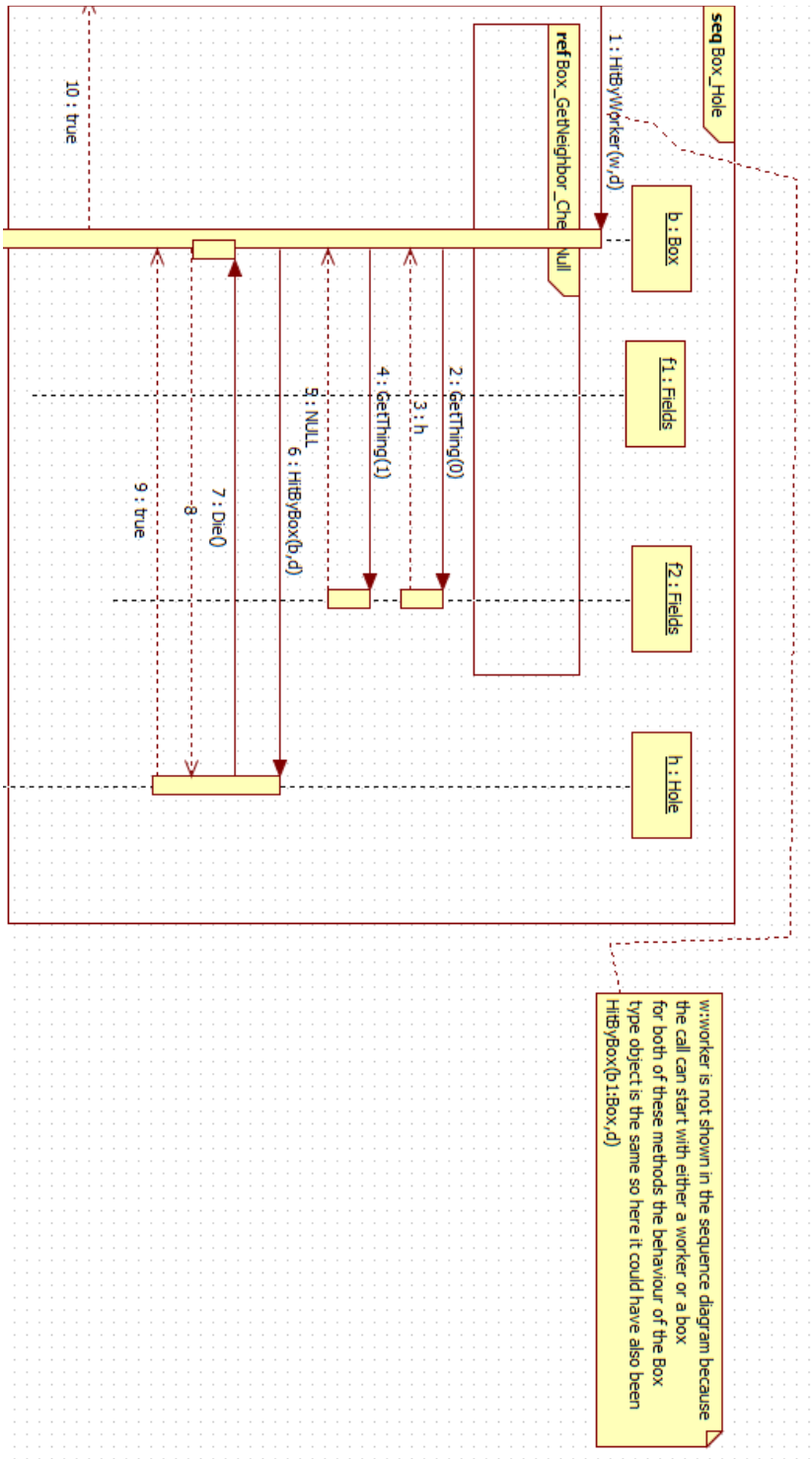
3.4.12 Box Worker

This Sequence Diagram shows when a Box is moved on a Worker.



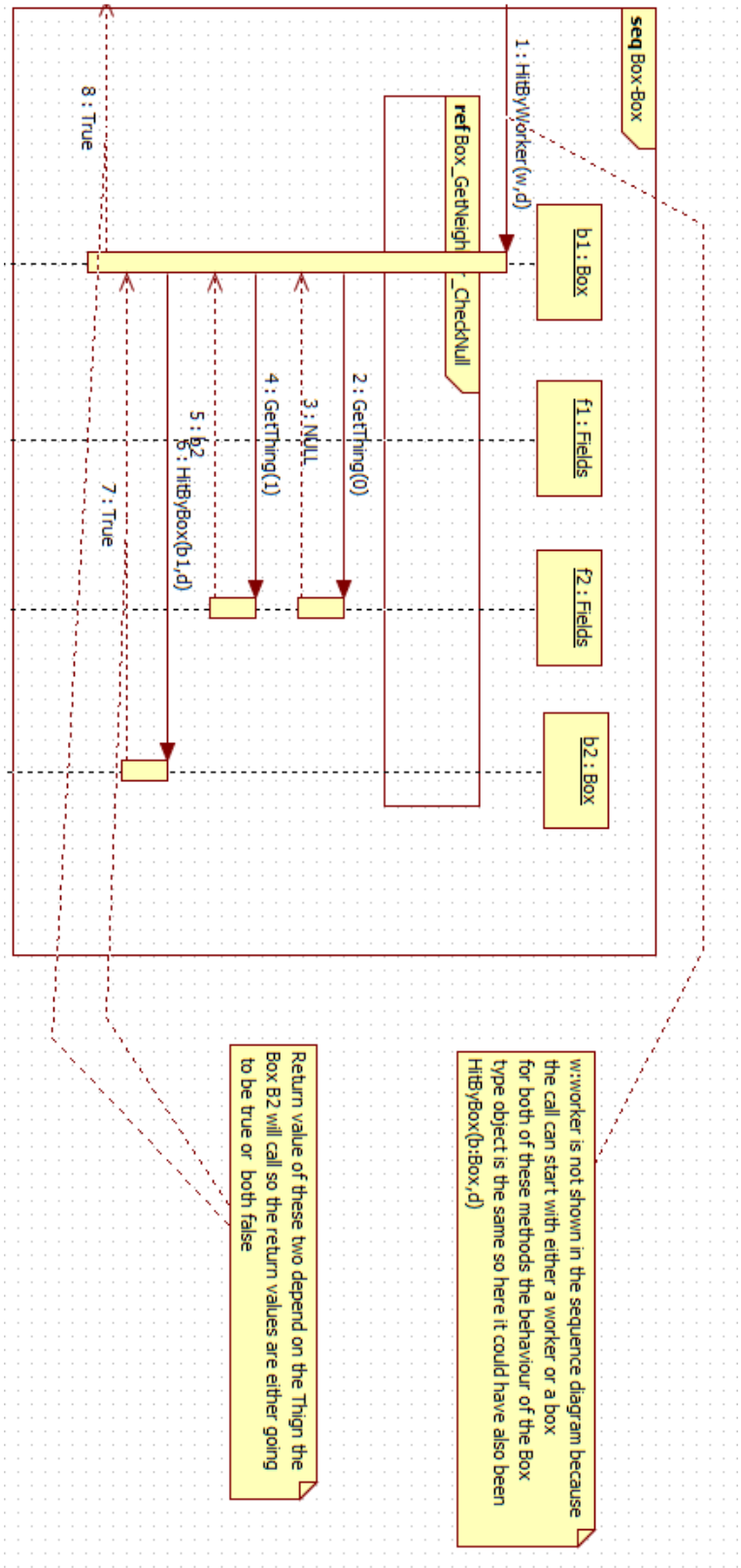
3.4.13 Box Hole

This Sequence Diagram shows when a Box is moved on a Hole.



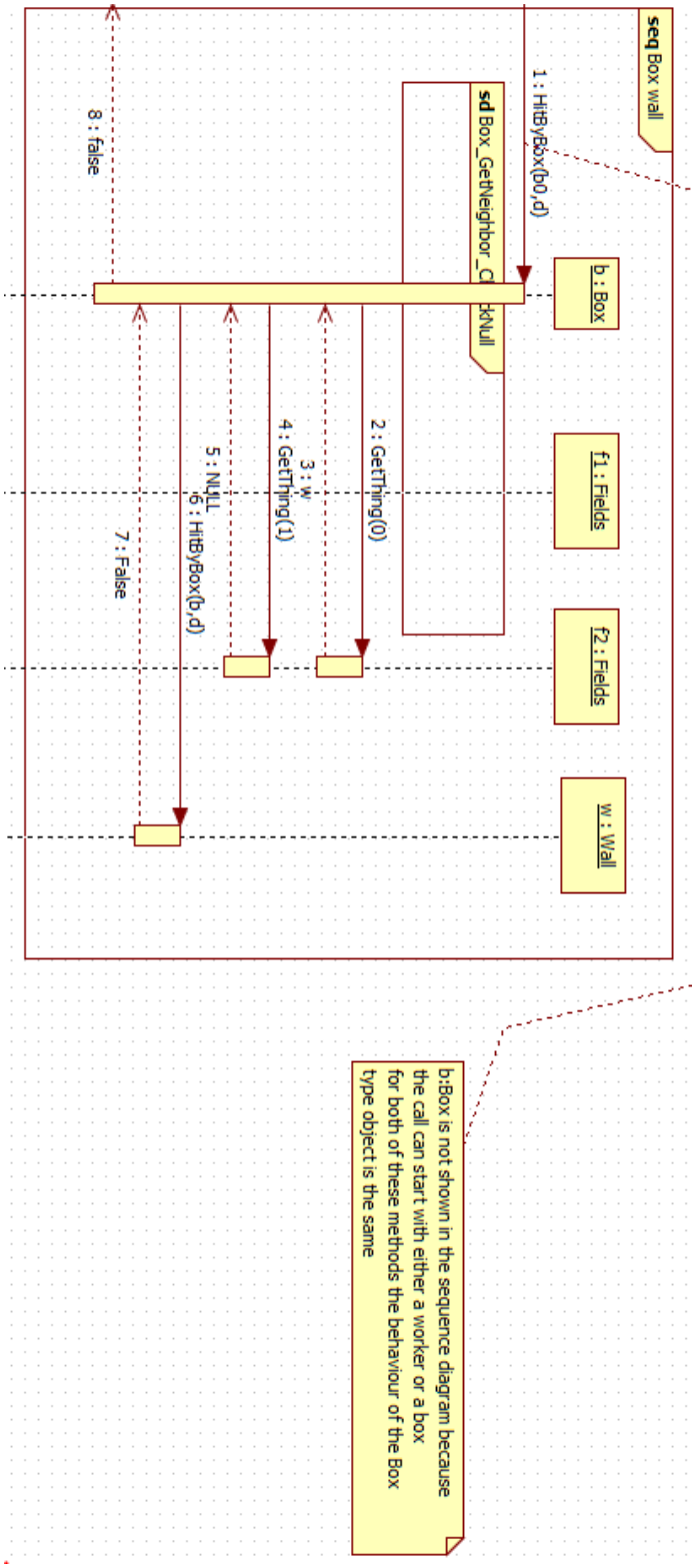
3.4.14 Box Box

This Sequence Diagram shows when a Box is moved on a Box.



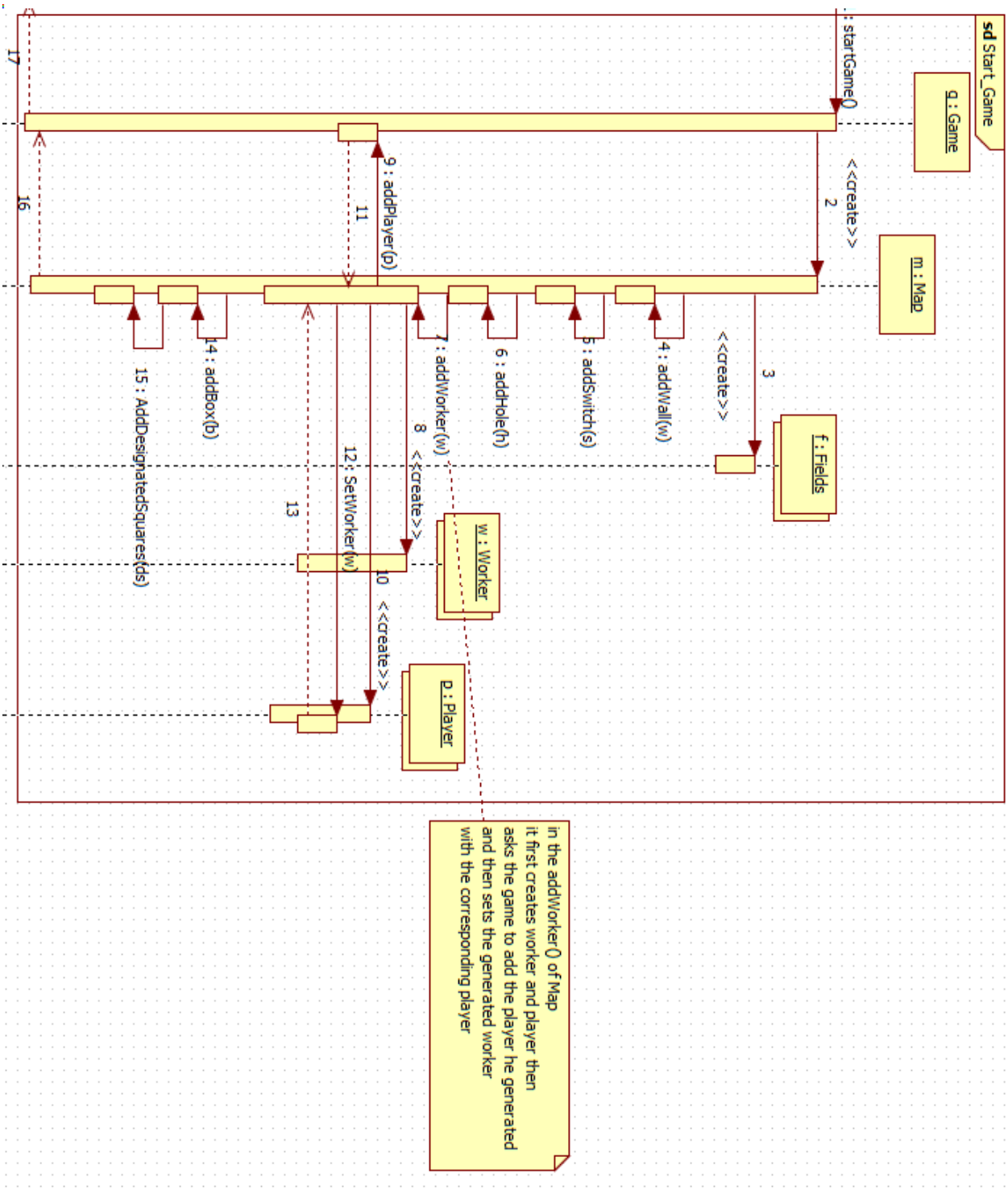
3.4.15 Box Wall

This Sequence Diagram shows when a Box is moved on a Wall.



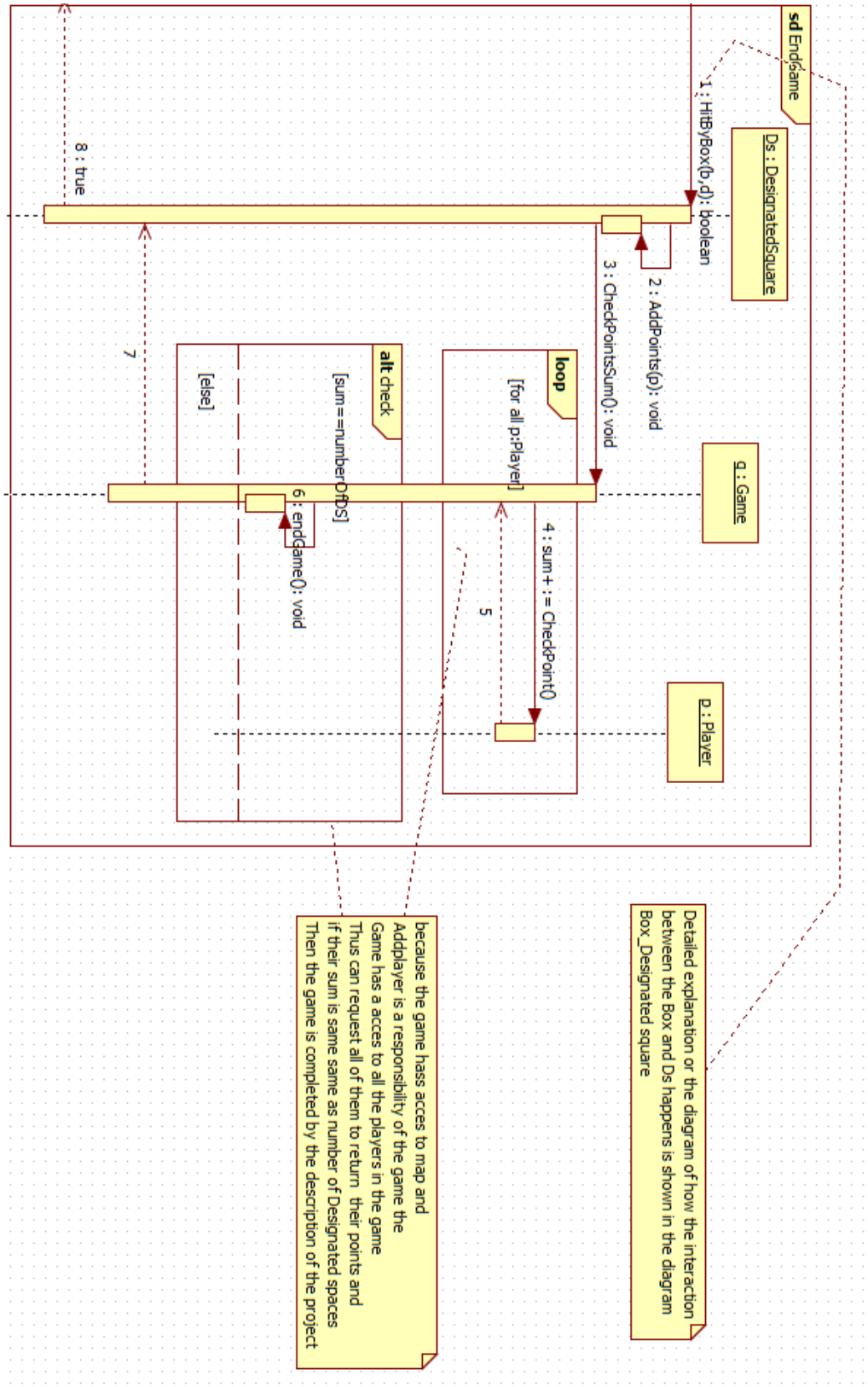
3.4.16 Start game

Note: We were not able to make the <<create>> properly.



3.4.17 End Game

This sequence diagram shows how the game ending happens.



[redacted]*Protocol*

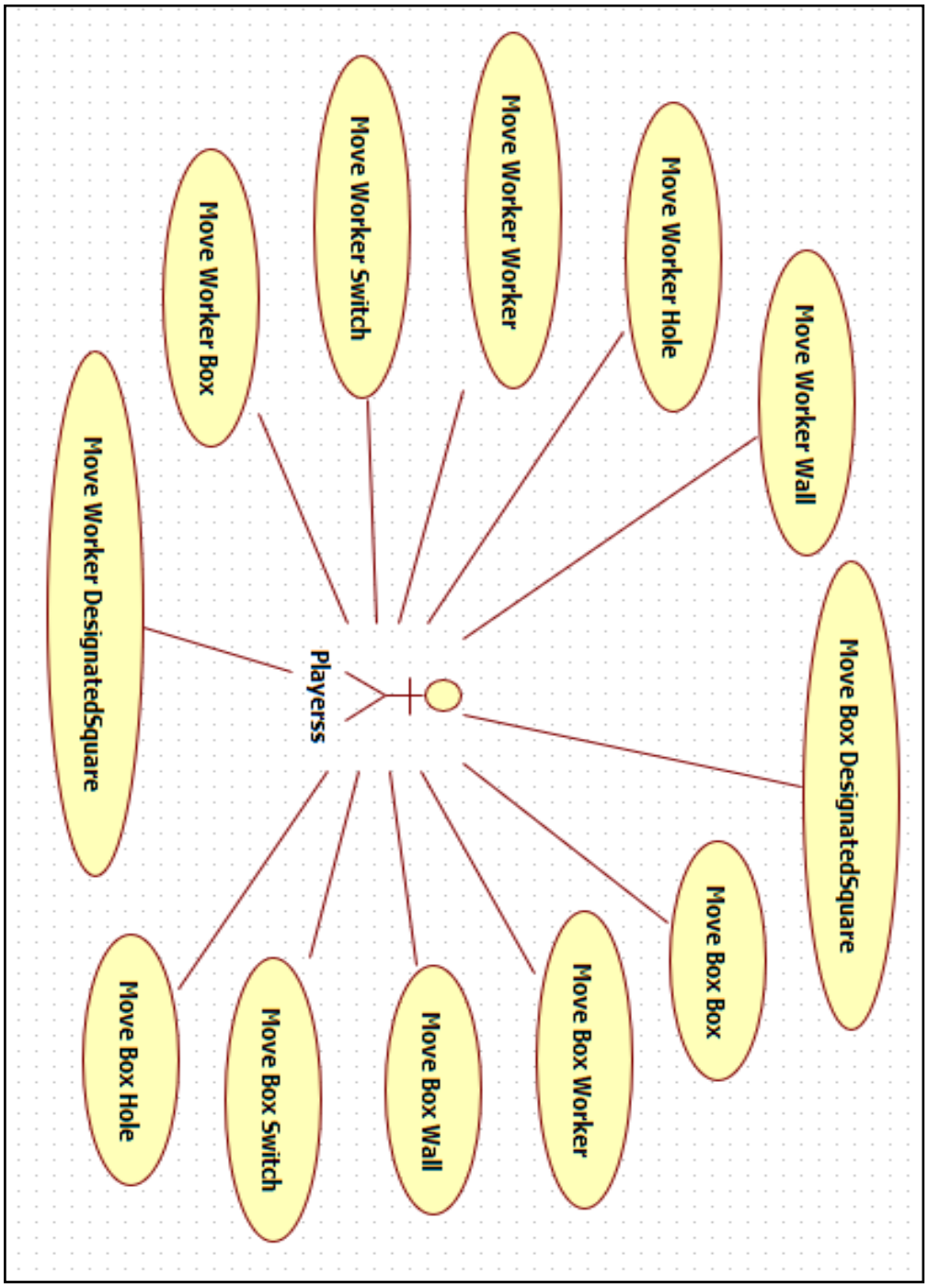
Start (date & time)	Duration (hours)	Performer(s) name	Activity description
[28.02.2018. 13:00]	*[5 hours]*	*All team's members*	[Meeting]
[01.03.2018. 14:00]	*[4 hours]*	*All team's members*	[Meeting]
[03.03.2018. 11:00]	*[8 hours]*	*All team's members*	[Meeting]
[04.03.2018. 11:00]	*[10 hours]*	*All team's members*	[Meeting]

5. Planning the skeleton

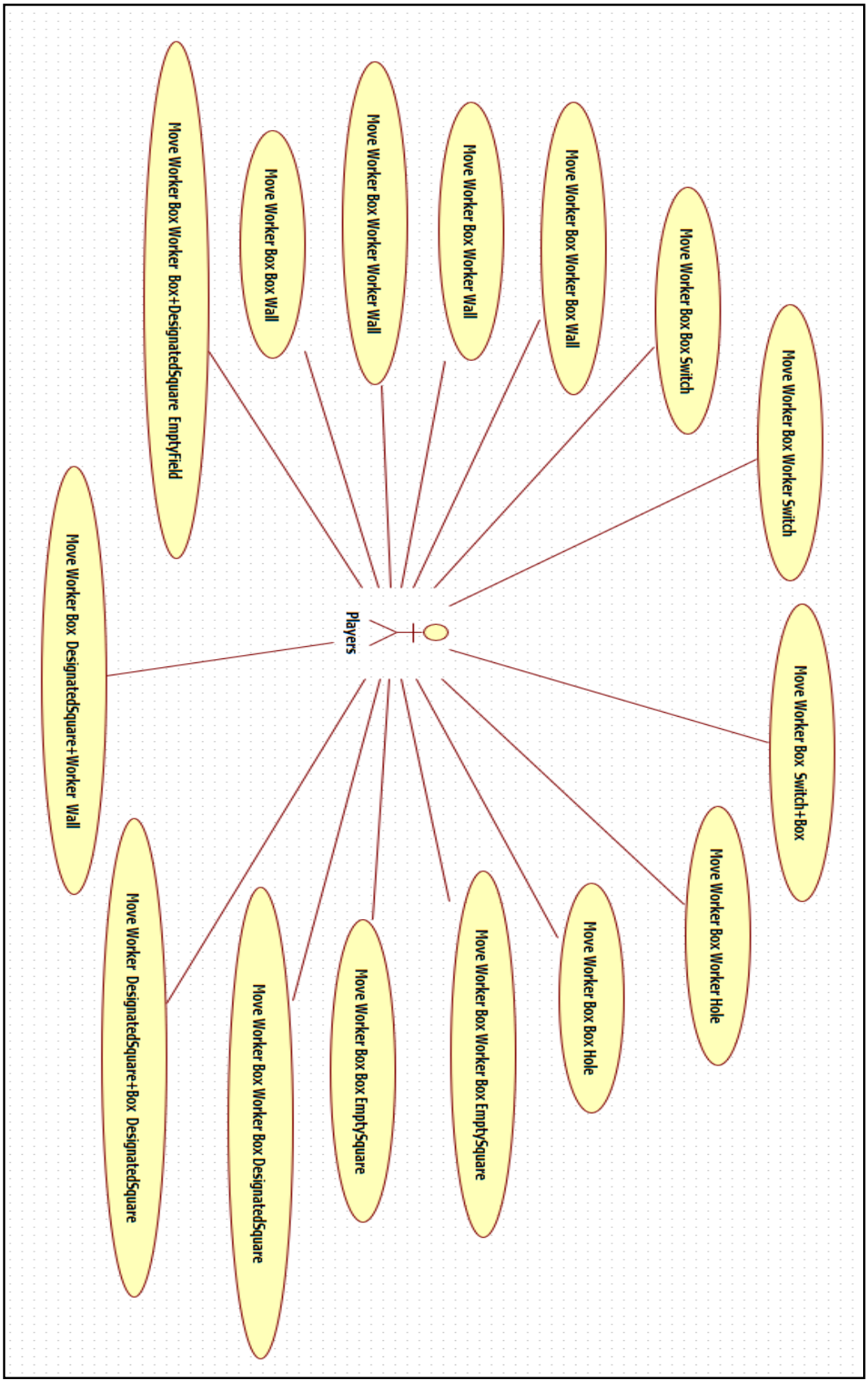
5.1 Real use-cases of the skeleton model

5.1.1 Use-case diagram

Basic Moves:



Complex Moves:



5.1.2 Use-case descriptions

Use-case name	Move Worker Box Worker Switch
Short textual description	The Worker pushes the Box which pushes the second Worker on a Switch
Actors	Player
Dialog, scenario	When a Worker pushes a Box on a Worker it tries to step on (HitByWorker) the things of the neighbouring Field in this case it is a Switch Then the Worker is moved on the Switch and it's status stays the same.

Use-case name	Move Worker Box Box Switch
Short textual description	The Worker pushes the Box that pushes the second Box on a Switch
Actors	Player
Dialog, scenario	When a Worker pushes a Box on another Box which will intail try to push(HitByBox) the Switch on the next field then the Box is moved on the Switch and it's inner state changes to on from off and informs the hole abouts it's state change through its getters and setters.

Use-case name	Move Worker Box Worker Box Wall
Short textual description	The Worker pushes the Box that pushes the second Worker on a Box that is pushed on a Wall
Actors	Player
Dialog, scenario	When a Worker pushes a Box on a Worker then the worker will try to push the Box in the same direction if uncessfull it will be crushed by the box trying to hit it.It will die.

Use-case name	Move Worker Box Worker Wall
Short textual description	The Worker pushes the Box that pushes the second Worker that is pushed on a Wall
Actors	Player
Dialog, scenario	When a Worker pushes a Box on a Worker then the worker will try to push the Box in the same direction if uncessfull it will be crushed by the box trying to hit it.It will die.

Use-case name	Move Worker Box Worker Worker Wall
Short textual description	When a Worker is pushed onto another Worker by a Box he dies and other entities are moved to their corresponding positions
Actors	Player
Dialog, scenario	When a Worker pushes a Box on a Worker i will try to move in the direction of the second Worker but because the return of the hitbyworker of any worker type entity is false the worker will die.The entites that came before the elliminated object are relocated.

Use-case name	Move Worker Box Box Wall
Short textual description	The Worker hits the Box that hits the second Box that tries to hit on a Wall
Actors	Player
Dialog, scenario	When a Worker pushes a Box on a second Box this will try to push onto the wall but the return value will be false thus the relocation will not happen for any of the enlisted entities.

Use-case name	Move Worker Box Worker (Box + DesignatedSquare (DS)) EmptyField
Short textual description	A worker pushes a box onto another worker which will push the Box out of the field that the DS is on.
Actors	Player
Dialog, scenario	When the second box is pushed out, the worker that initiated the move will loose points.

Use-case name	Move Worker Box (DesignatedSquare + Worker) Wall
Short textual description	A Worker pushes a box on the second Worker
Actors	Player
Dialog, scenario	The designated square acts as a normal square, therefor the Wall blocks the second Worker from moving, and he dies.

Use-case name	Move Worker (DesignatedSquare + Box) DesignatedSquare
Short textual description	The worker pushes the box from a DS to another DS
Actors	Player
Dialog, scenario	The Player looses a point when it's pushed out, and then gets a point when it reaches the second DS.

Use-case name	Move Worker Box Worker Box DesignatedSquare
Short textual description	The Worker pushes a box, which pushs a worker and another box into a DS.
Actors	Player
Dialog, scenario	When the Worker pushes the Box that pushes the second Worker that pushes the second Box on the DesignatedSquare the first Worker is the one that gets the point and every object moves on the next field on the given direction.

Use-case name	Move Worker Box Box Worker EmptySquare
Short textual description	The Worker pushes the Box which pushes the second Box on a Worker then an EmptySquare.
Actors	Player
Dialog, scenario	When a Worker pushes a Box on a Box it checks the next field where it finds a worker, so every object moves on the next field in the given direction

Use-case name	Move Worker Box Worker Box EmptySquare
Short textual description	The Worker pushes the Box that pushes the second Worker that pushes the second Box on an EmptySquare.
Actors	Player
Dialog, scenario	When a Worker pushes a Box that is then pushed on a second Worker that is pushed on the second Box that is moved on the EmptySquare every object moves on the next field in the given direction.

Use-case name	Move Worker Box Box Hole
Short textual description	The Worker pushes the Box that pushes the second Box that that is pushed in a Hole
Actors	Player
Dialog, scenario	When a Worker pushes a Box on a Box which is then pushed into a Hole, the second Box Dies.

Use-case name	Move Worker Box Worker Hole
Short textual description	The Worker pushes the Box that Pushes the second Worker that is pushed in a Hole
Actors	Player
Dialog, scenario	When a Worker pushes a Box on a Worker which is then pushed into a Hole, the second Worker Dies.

Use-case name	Move Worker Box (Switch + Box)
Short textual description	A Worker pushes a Box on another Box which is on a Switch
Actors	Player
Dialog, scenario	The status of the switch is ON since it has a box, when the box is pushed out by the first box, it goes OFF and then the first box steps on the switch and turns it ON again.

Use-case name	Move Worker Wall
Short textual description	Worker tries to move and there is a Wall
Actors	Player
Dialog, scenario	The wall stops the move of the Worker

Use-case name	Move Worker Hole
Short textual description	Worker moves to a Hole
Actors	Player
Dialog, scenario	When Worker steps on Hole the Worker dies

Use-case name	Move Worker Worker
Short textual description	A Worker pushs a Worker
Actors	Player
Dialog, scenario	A Worker cannot push another Worker directly

Use-case name	Move Worker Switch
Short textual description	a Worker tries to move and there is a Switch
Actors	Player

Dialog, scenario	The switch does not change it's state when the worker steps on it.

Use-case name	Move Worker Box
Short textual description	Worker moves the Box to an empty field
Actors	Player
Dialog, scenario	The Box is hit by the worker

Use-case name	Move Worker DesignateSquare
Short textual description	Worker moves into a DS
Actors	Player
Dialog, scenario	When the worker moves into the DS, nothing happens.

Use-case name	Move Box Hole
Short textual description	When a Box is moved on a Hole
Actors	Player
Dialog, scenario	When the Box steps on a Hole, it dies.

Use-case name	Move Box Switch
Short textual description	A Worker pushes a Box or Boxes onto the Switch the inner state of the corresponding Switch and the associated Hole will be changed.
Actors	Player
Dialog, scenario	When the Box is on the Switch it turns the Switch ON

Use-case name	Move Box Wall
Short textual description	When a Box is moved on a Wall
Actors	Player
Dialog, scenario	The box doesnt move because the wall blocks it.

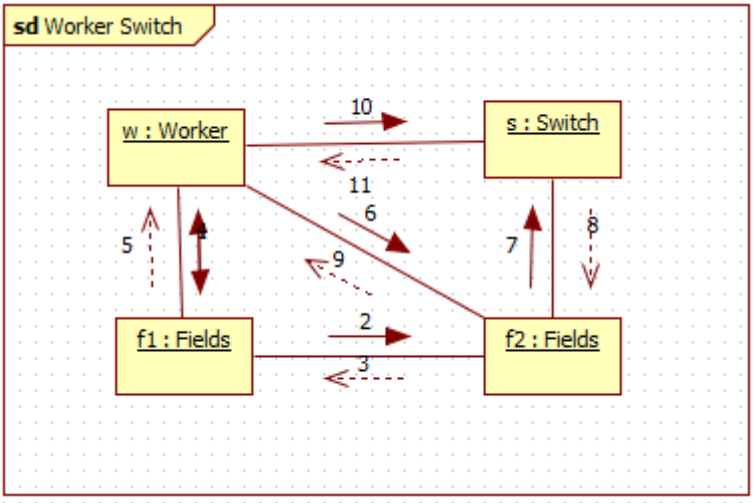
Use-case name	Move Box Worker
Short textual description	When a Box is moved on a Worker
Actors	Player
Dialog, scenario	The Box pushes the Worker, what happens depends on the next field.

Use-case name	Move Box Box
Short textual description	When a Box is moved on a Box
Actors	Player
Dialog, scenario	The Box os pushed, if it's an empty field, both of them move to the next field.

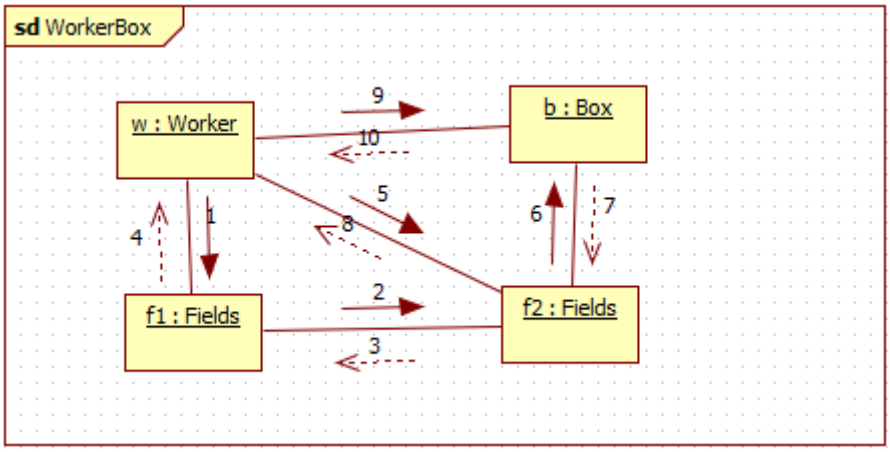
Use-case name	Move Box DesignatedSquare
Short textual description	Box is moved on a DS
Actors	Player
Dialog, scenario	When the Box steps on the DS, the Worker who initiated the move gets a point.

5.2 Architecture

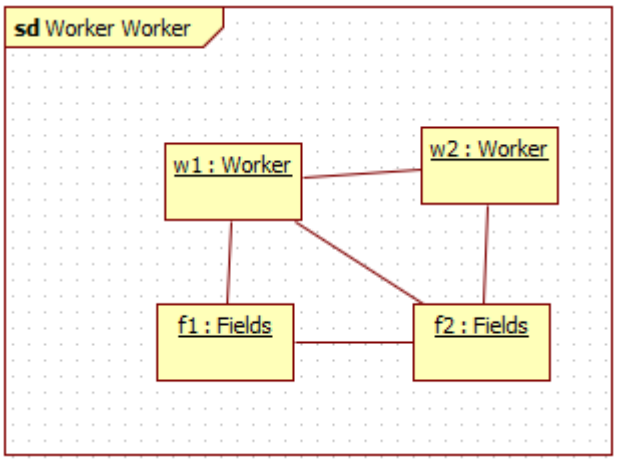
5.2.1 Worker Switch



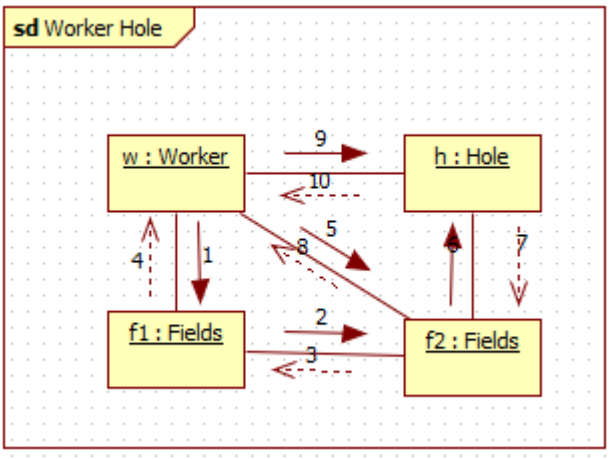
5.2.2 Worker Box



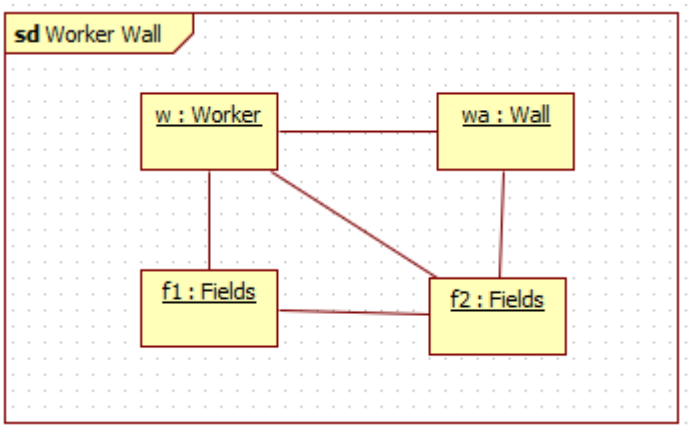
5.2.3 Worker Worker



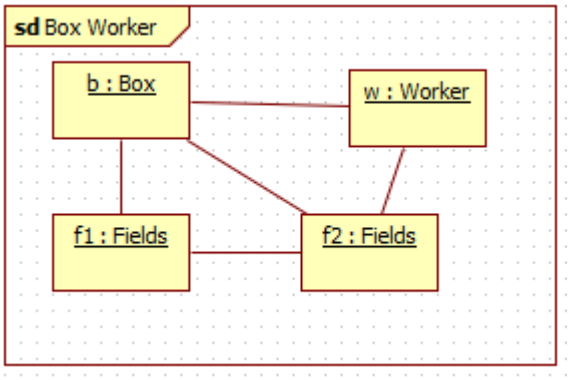
5.2.4 Worker Hole



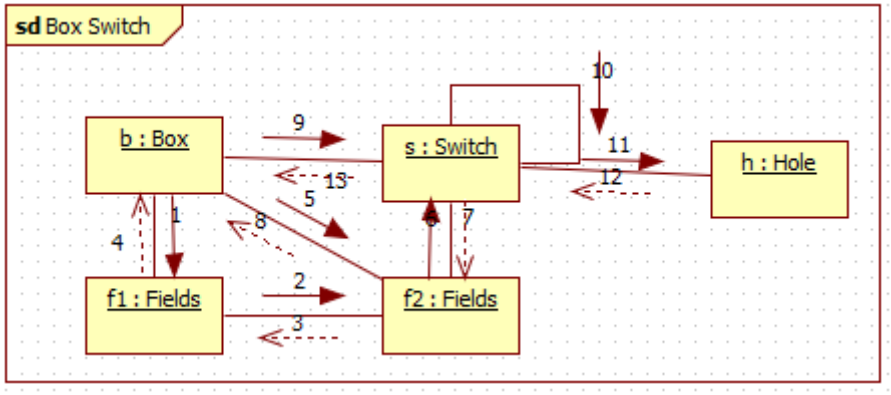
5.2.5 Worker Wall



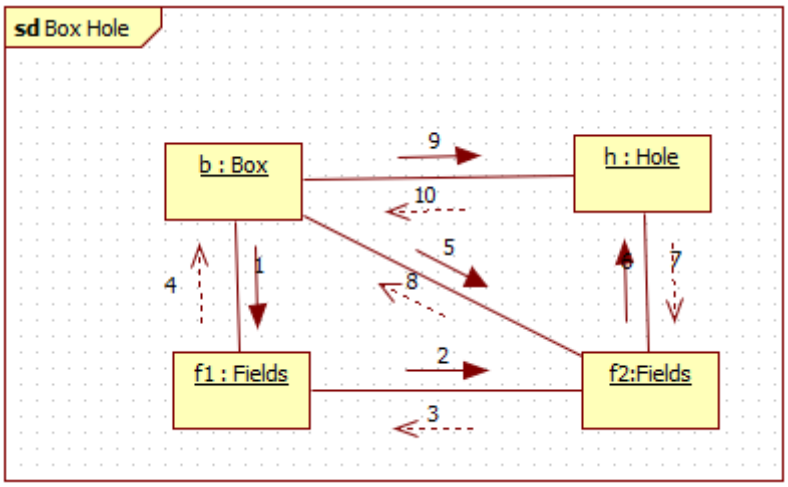
5.2.6 Box Worker



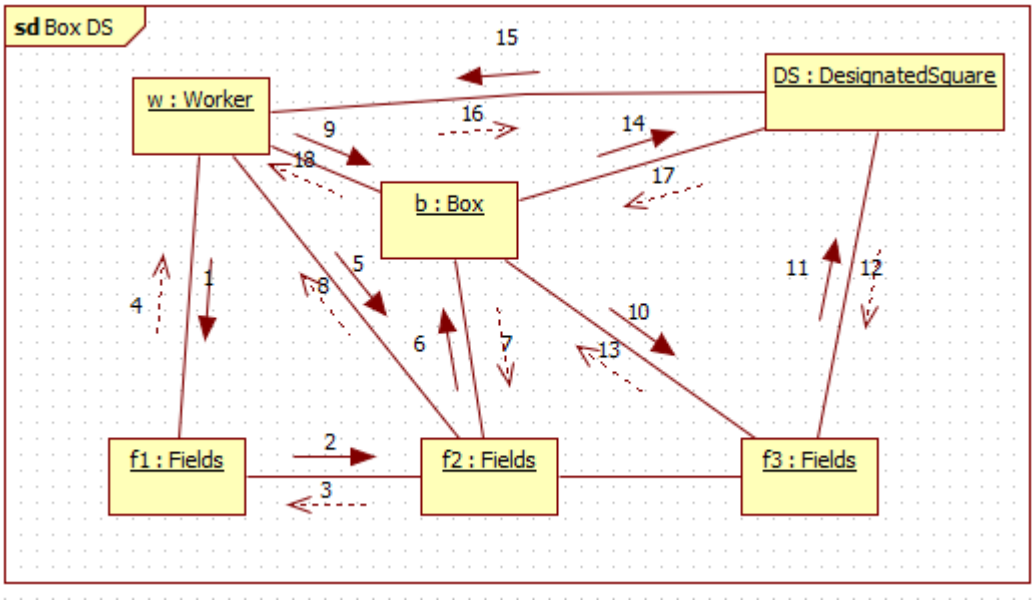
5.2.7 Box Switch



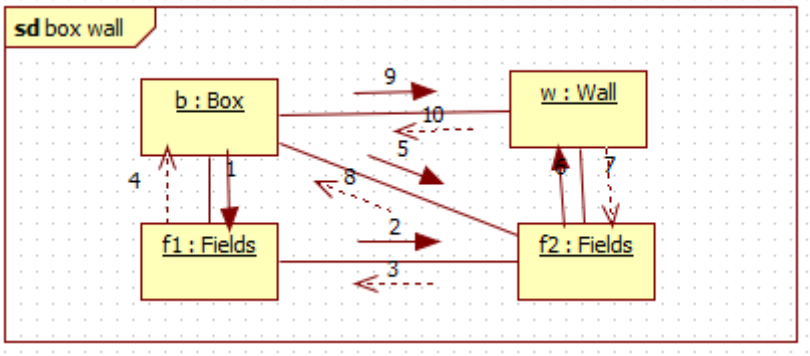
5.2.8 Box Hole



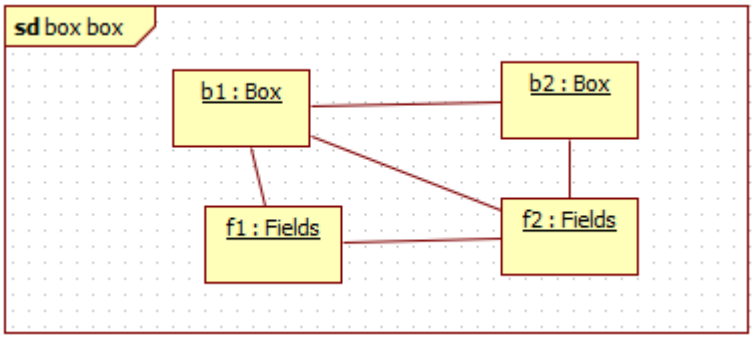
5.2.9 Box DesignatedSquare



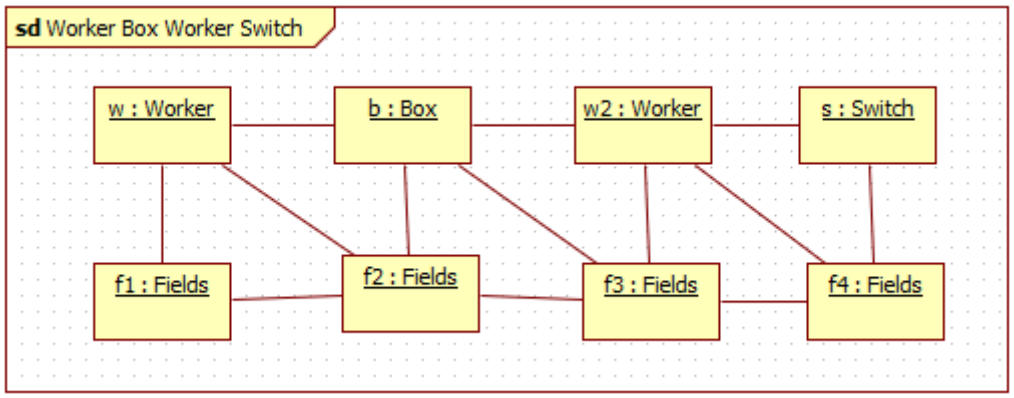
5.2.10 Box Wall



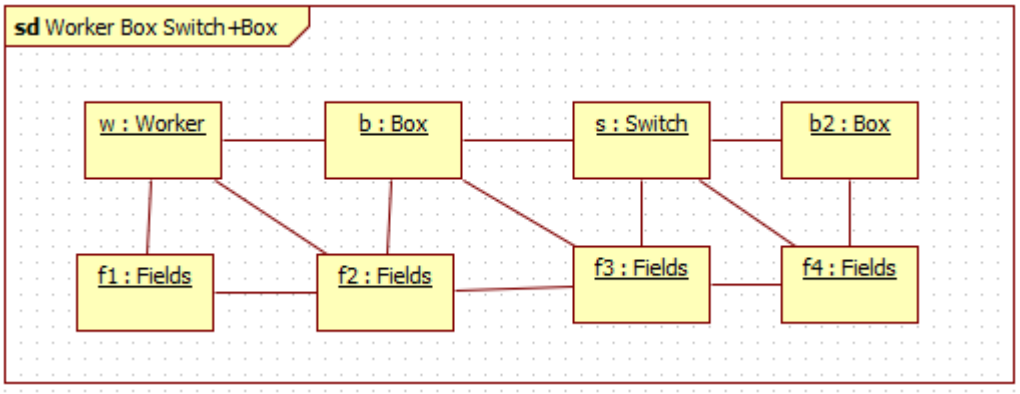
5.2.11 Box Box



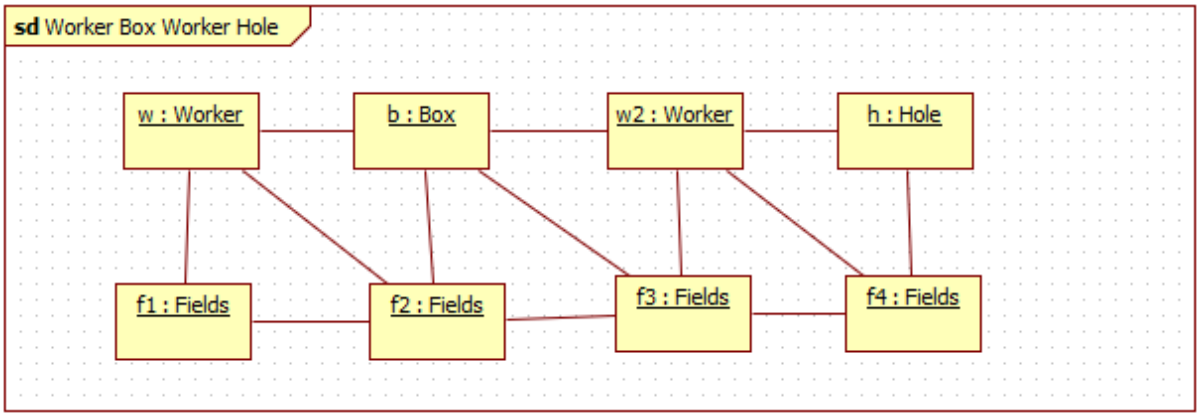
5.2.12 Worker Box Worker Switch



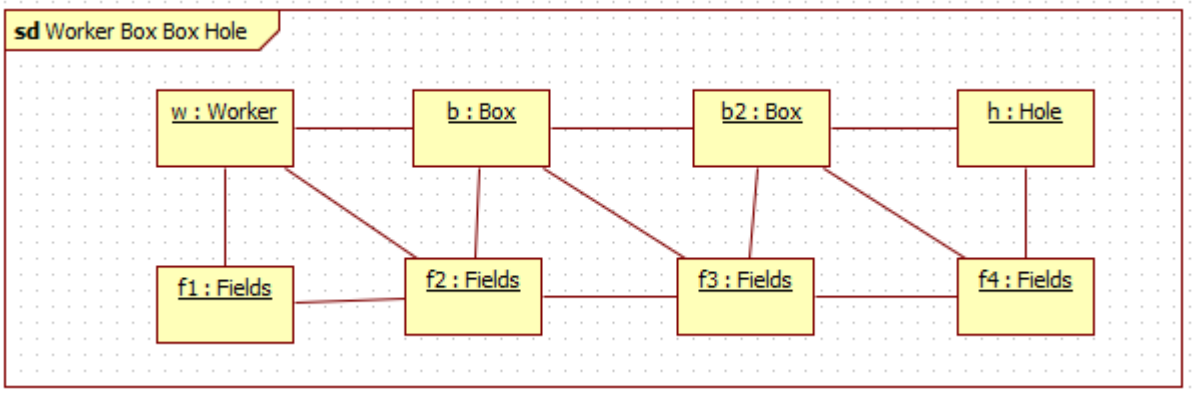
5.2.13 Worker Box (Switch+Box)



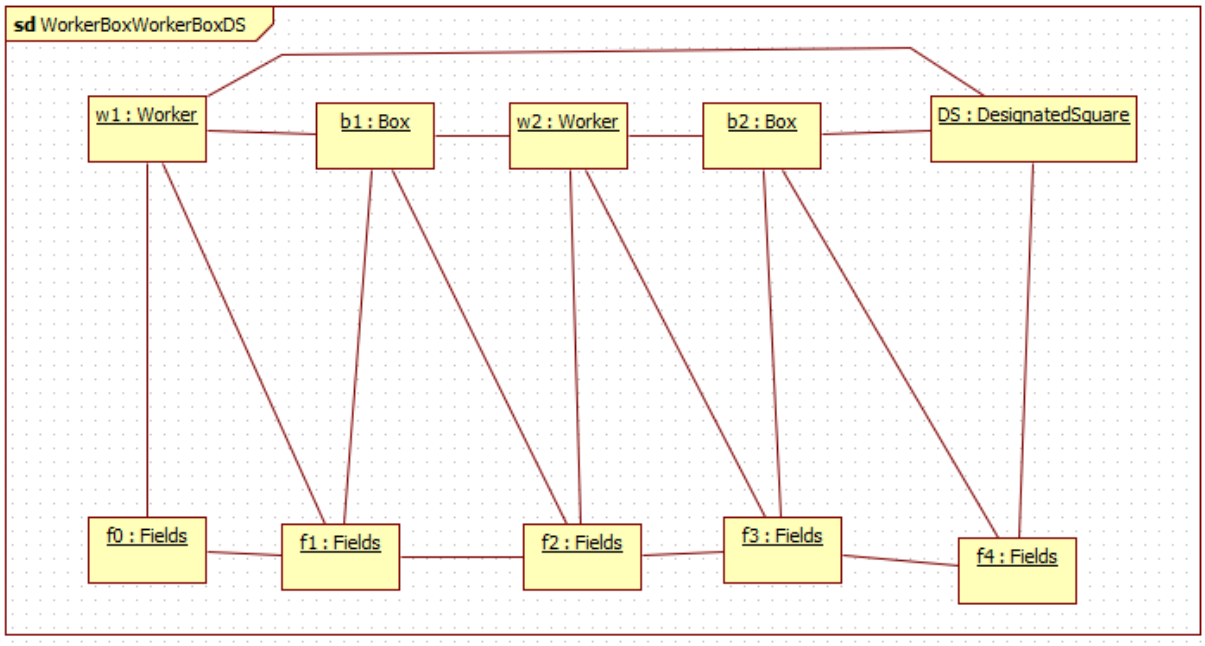
5.2.14 Worker Box Worker Hole



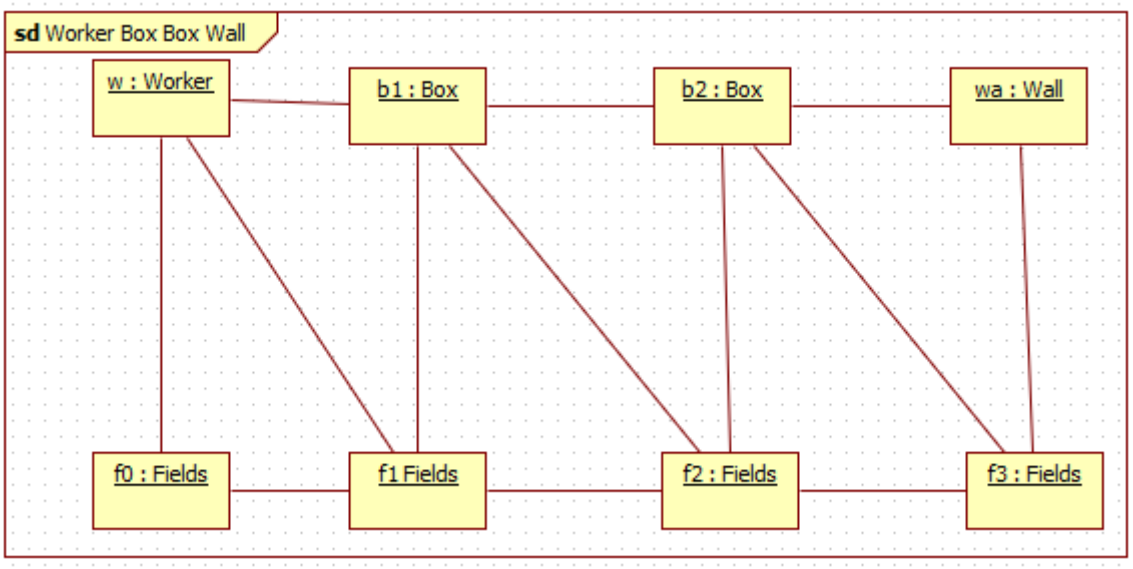
5.2.15 Worker Box Box Hole



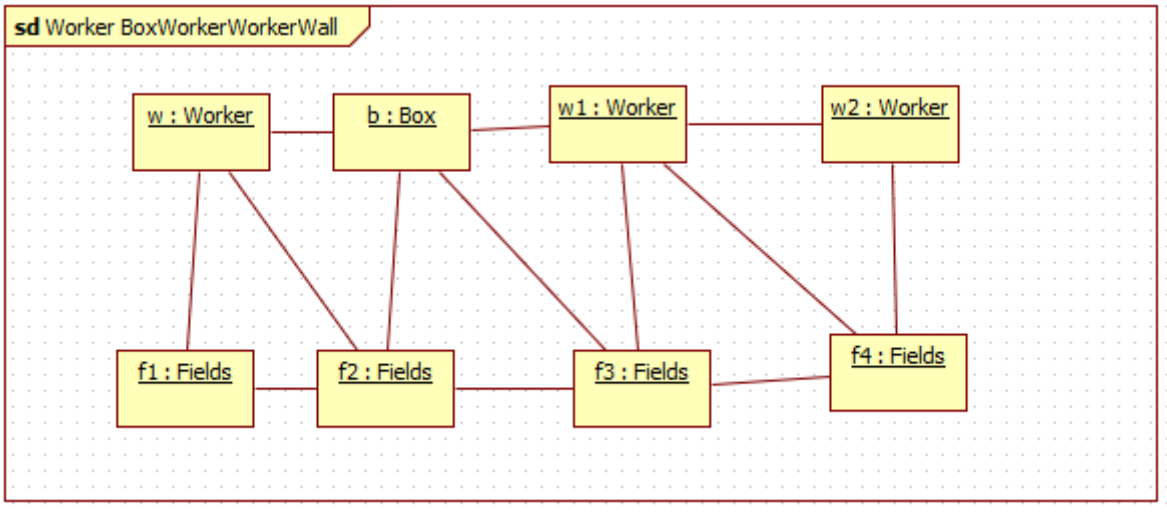
5.2.16 Worker Box Worker Box DesignatedSquare



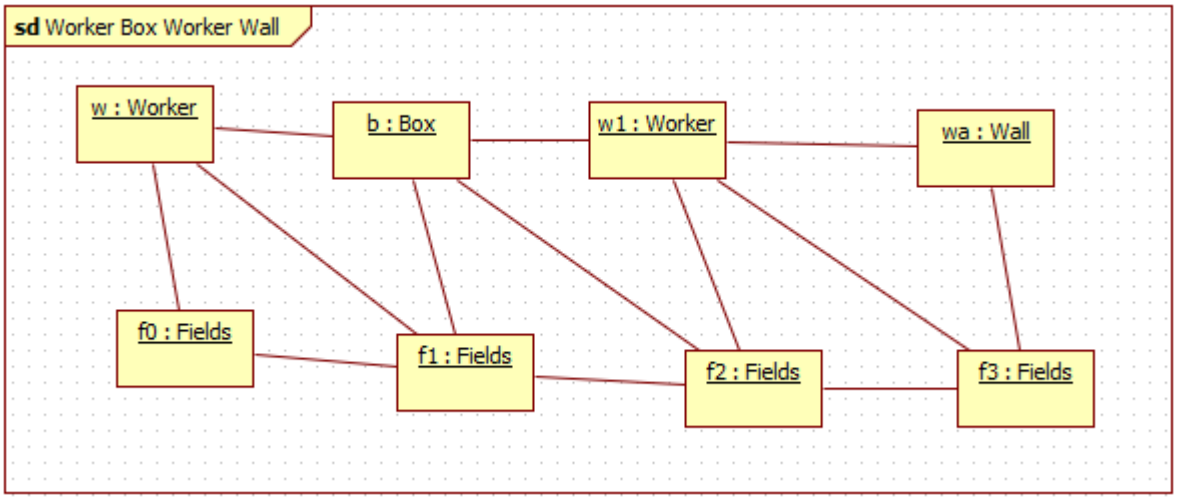
5.2.17 Worker Box Box Wall



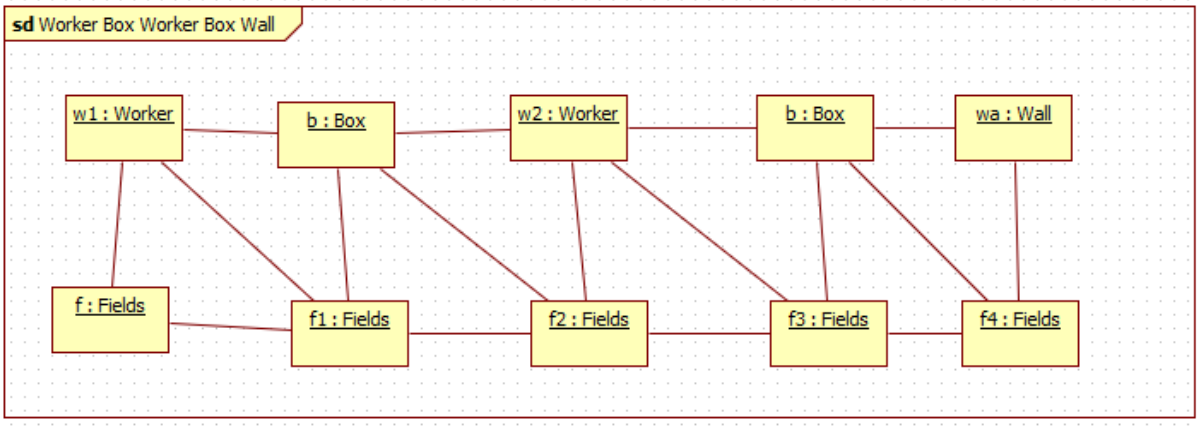
5.2.18 Worker Box Worker Worker Wall



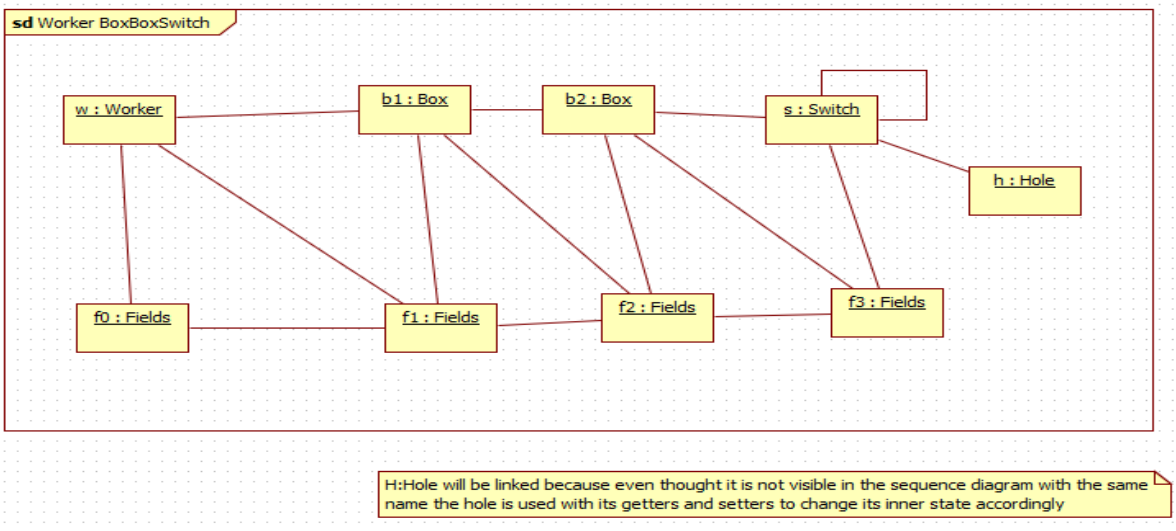
5.2.19 Worker Box Worker Wall



5.2.20 Worker Box Worker Box Wall



5.2.21 Worker Box Box Switch



5.3 Plans of the skeleton's UI, dialogs

```
>   Move Worker(w1) Switch(s)

>   =>f1.GetNeighbour(d)
>   =>f2.GetThing()
>   <= Things[s]
>   =>s.HitByWorker(w1,d)
>   <=True
>   =>f2.Accept(w1)
>   =>f1.Remove(w1)
>   The Worker Stepped on the Switch,
>   and it kept the same state.
```

```
>   Move Worker(w) Box(b)
>
>   =>f1.GetNeighbour(d)
>   =>f2.GetThing()
>   <= Things[b]
>   =>b.HitByWorker(w,d)
>   <=True
>   =>f2.Accept(w)
>   =>f1.Remove(w)
>   The Worker pushed the Box.
```

```
>   Move Worker(w1) Worker(w2)
>
>   =>f1.GetNeighbour(d)
>   =>f2.GetThing()
>   <= Things[w2]
>   =>w2.HitByWorker(w1,d)
>   <=False
>   The two workers keep their position.
```

```
>  Move Worker(w) Hole(h)
>
>  =>f1.GetNeighbour(d)
>  =>f2.GetThing()
>  <= Things[h]
>  =>h.HitByWorker(w,d)
>  =>w.Die()
>  <=True
>  The worker steps on the hole and dies.
```

```
>   Move Worker(w) Wall(wa)
>
>   =>f1.GetNeighbour(d)
>   <=f2
>   =>f2.GetThing()
>   <= Things[wa]
>   =>wa.HitByWorker(w,d)
>   <=False
>   The worker move is stopped by the Wall.
```

```
>   Move Box(b) Worker(w)
>
>   =>f1.GetNeighbour(d)
>   <=f2
>   =>f2.GetThing()
>   <= Things[w]
>   =>w.HitByBox(b,d)
>   <=True
>   The Worker is Hit by the Box,
>   depending on the next field,
>   he dies or moves.
```

```
>   Move Box(b) Switch(s)
>
>   =>f1.GetNeighbour(d)
>   <=f2
>   =>f2.GetThing()
>   <= Things[s]
>   =>s.HitByBox(b,d)
>   =>s.GetHole()
>   =>h.SetHoleStatus(true)
>   <=True
>   The Box is moved on the switch,
>   and it's state is changed.
```

```
>   Move Box(b) Hole(h)
>
>   =>f1.GetNeighbour(d)
>   <=f2
>   =>f2.GetThing()
>   <= Things[h]
>   =>s.HitByBox(b,d)
>   =>s.GetHole()
>   =>h.SetHoleStatus(true)
>   <=True
>   The Box is moved on the hole,
>   the box dies.
```

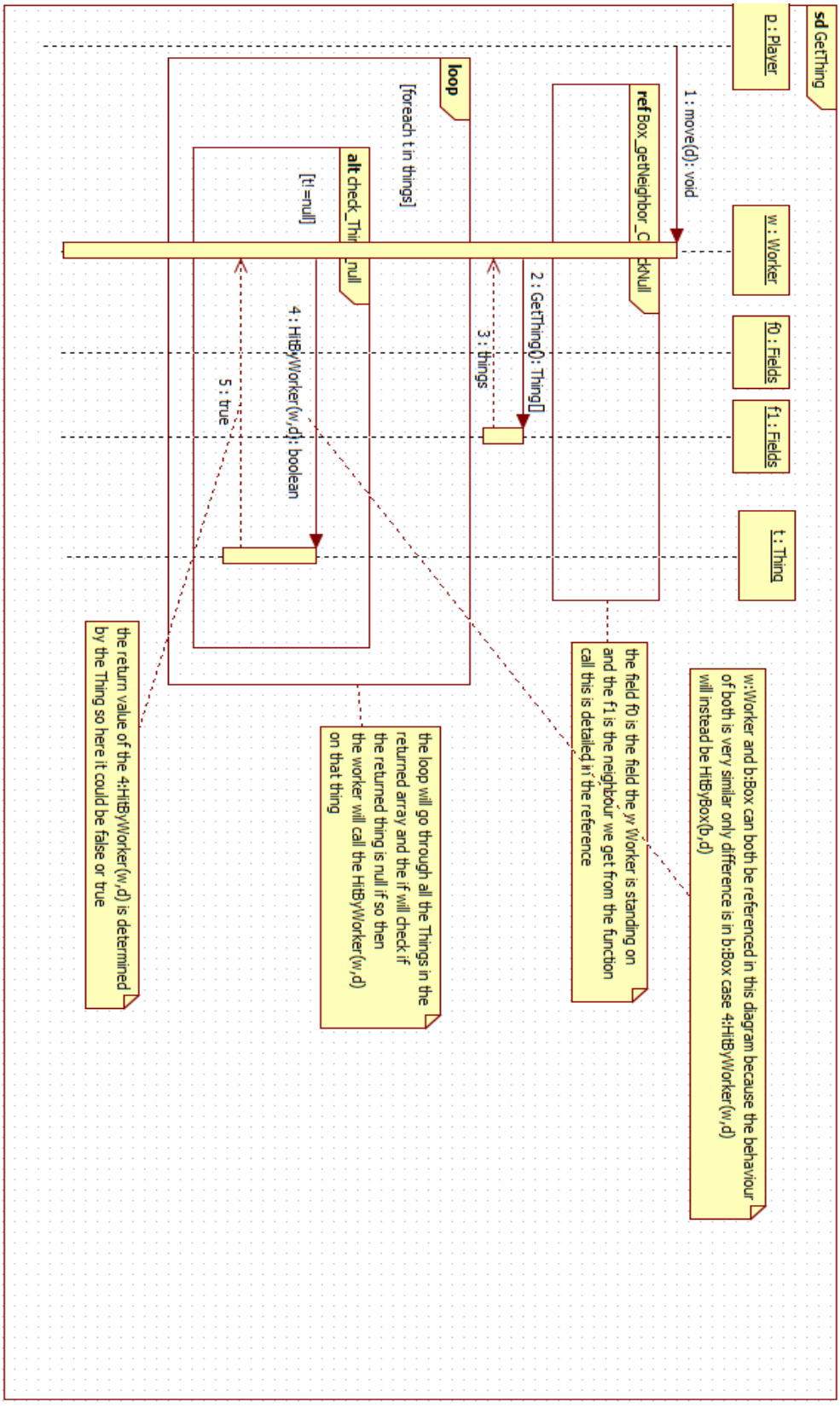
```
>   Move Box(b) DesignatedSquare(ds)
>
>   =>f1.GetNeighbour(d)
>   <=f2
>   =>f2.GetThing()
>   <= Things[ds]
>   =>ds.HitByBox(b,d)
>   =>GetPlayer()
>   <= player
>   =>ds.AddPoints(player)
>   <=True
>   The Box is moved on the DS,
>   the player who initiated the move
>   gets the points.
```

```
>   Move Box(b) Wall(wa)
>
>   =>f1.GetNeighbour(d)
>   <=f2
>   =>f2.GetThing()
>   <= Things[wa]
>   =>wa.HitByBox(b,d)
>   <=False
>   The Wall blocks the move.
```

```
>   Move Box(b1) Box(b2)
>
>   =>f1.GetNeighbour(d)
>   <=f2
>   =>f2.GetThing()
>   <= Things[b2]
>   =>b2.HitByBox(b1,d)
>   <=True
>   The Box is pushed on the other box,
>   each of them moves to the next field.
```

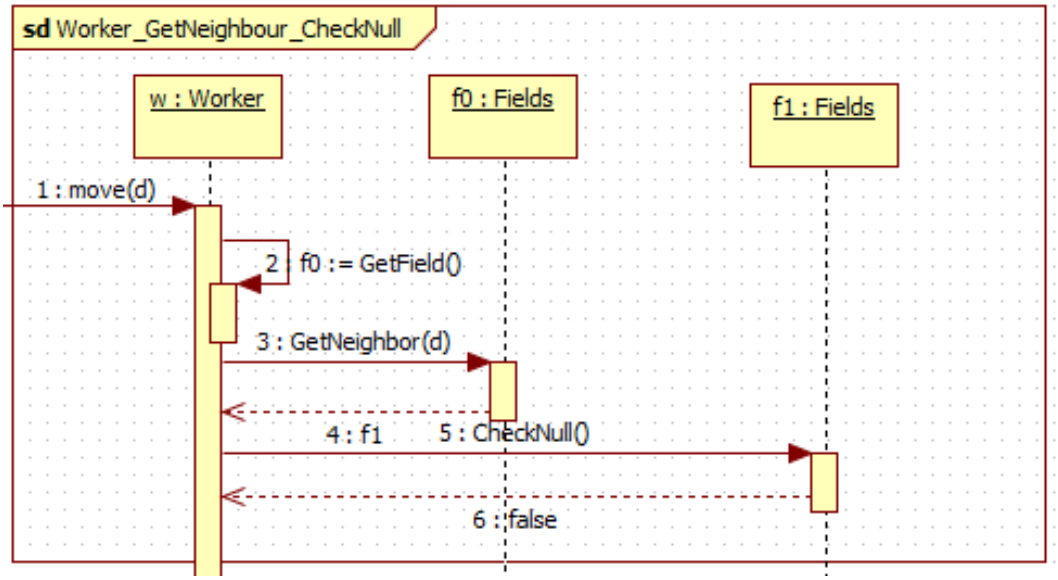
5.4 Detailed sequence diagrams for internal activities

Reference sequence diagrams: **1: GetThing() : Thing[]**

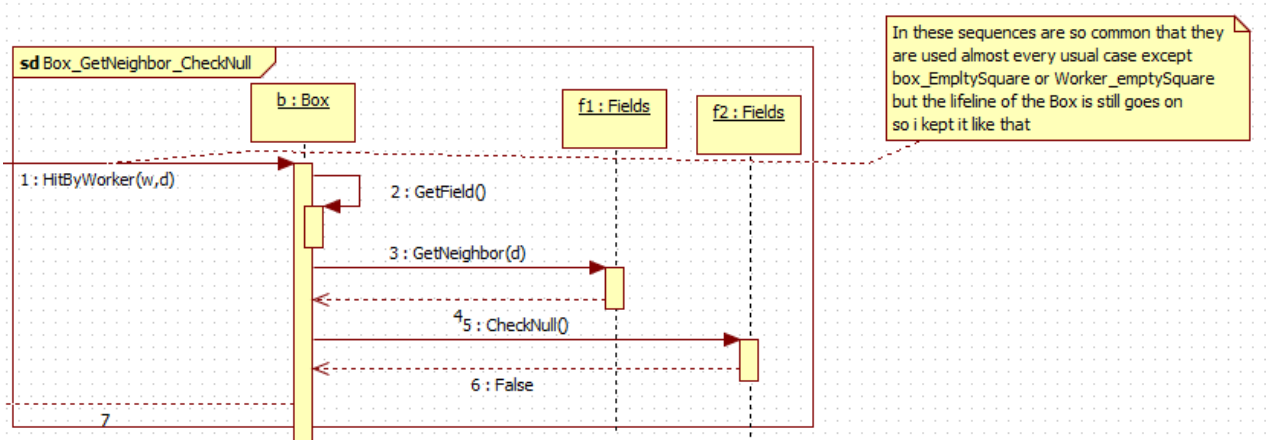


Important note on the diagrams mainly involving the worker the Remove and Accept have been omitted .They are present at all diagrams which have true as return value from the corresponding HitbyWorker request.

2- Worker GetNeighbour CheckNull:

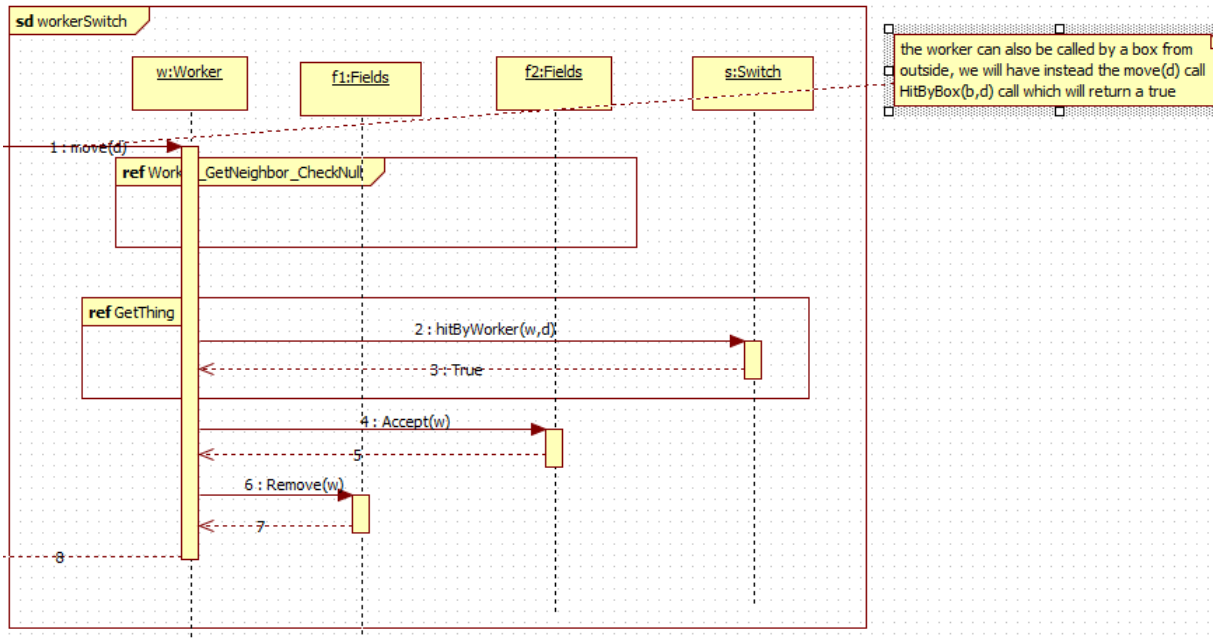


3- Box GetNeighbour CheckNull:

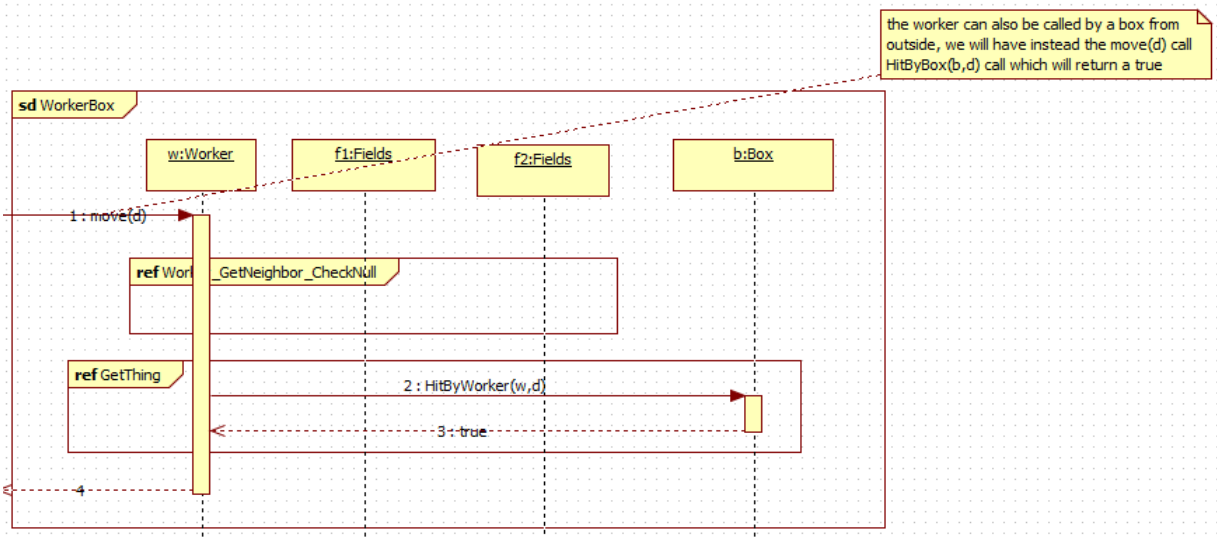


Basic Moves:

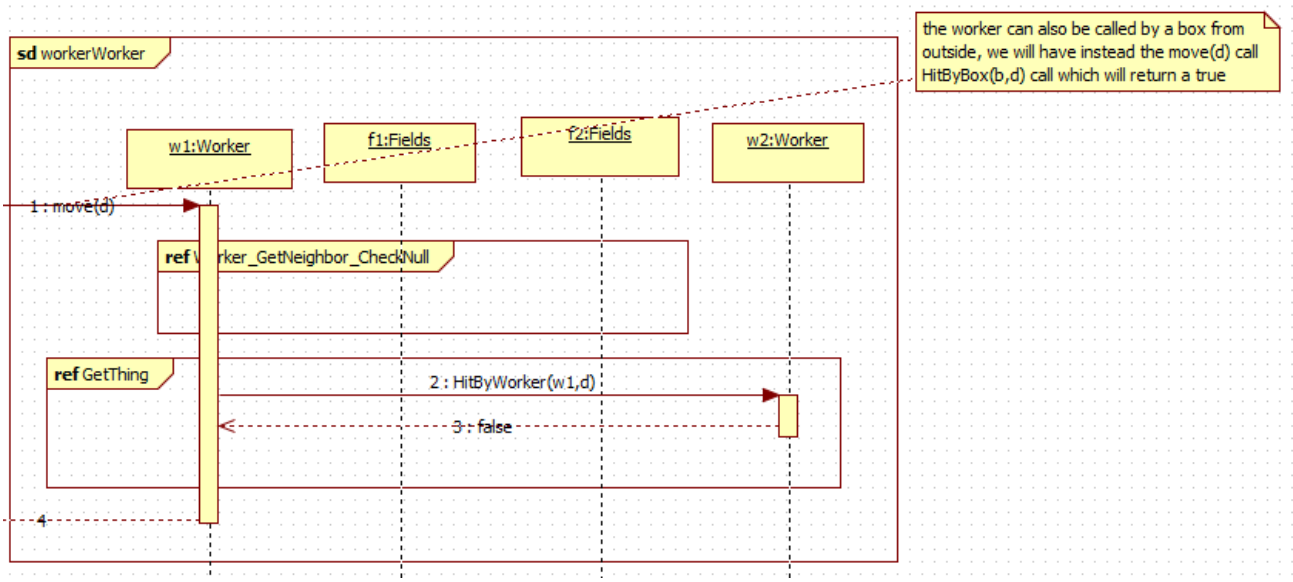
5.4.1 Move Worker Switch : Switch unchanged



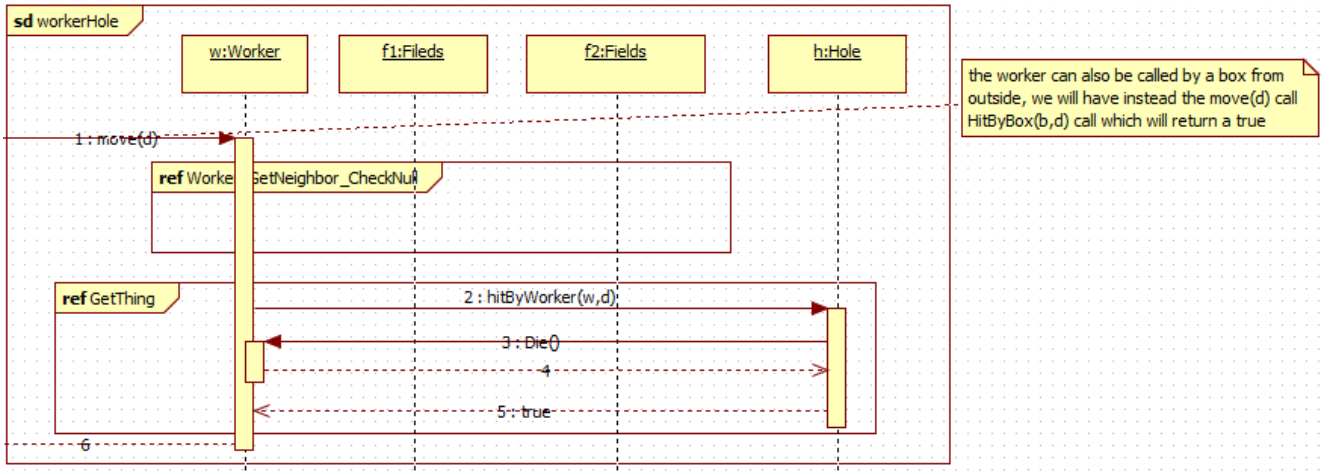
5.4.2 Move Worker Box: Worker and Box moved



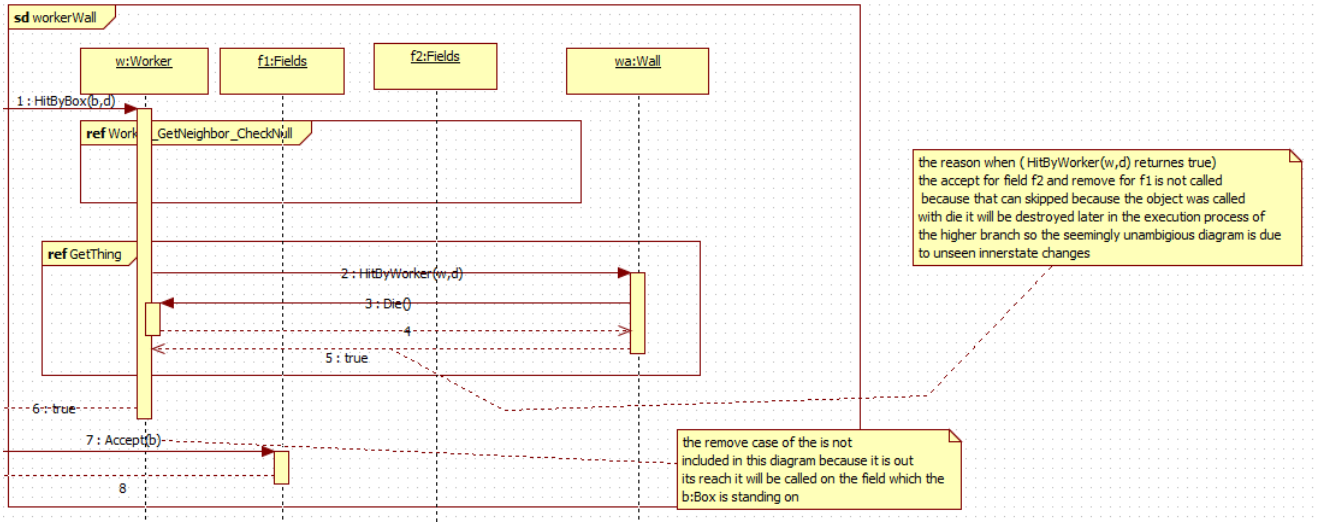
5.4.3 Move Worker Worker: Worker doesn't step on another worker.



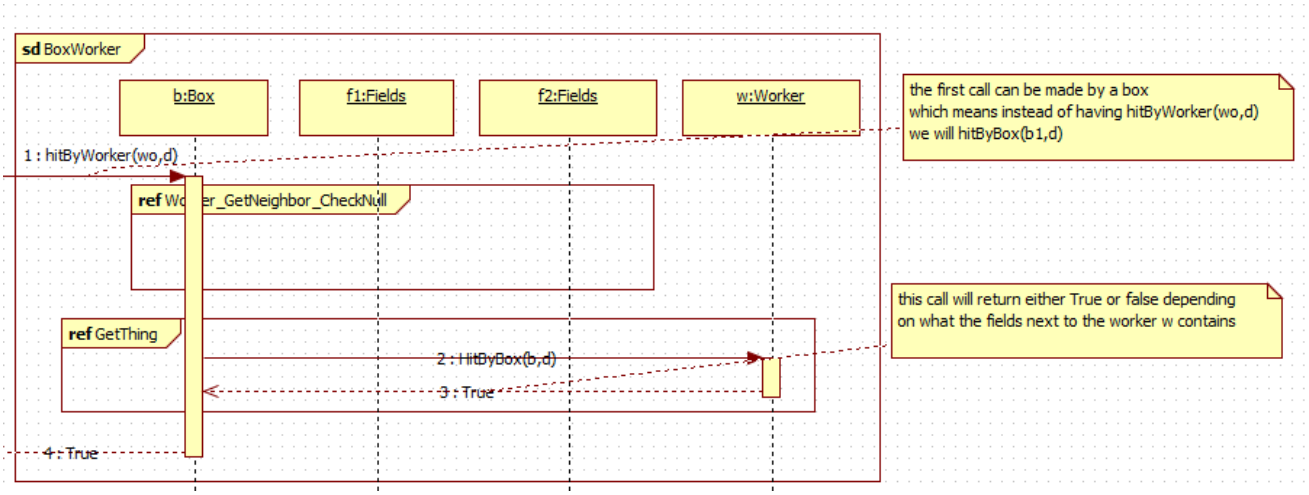
5.4.4 Move Worker Hole: Worker dies



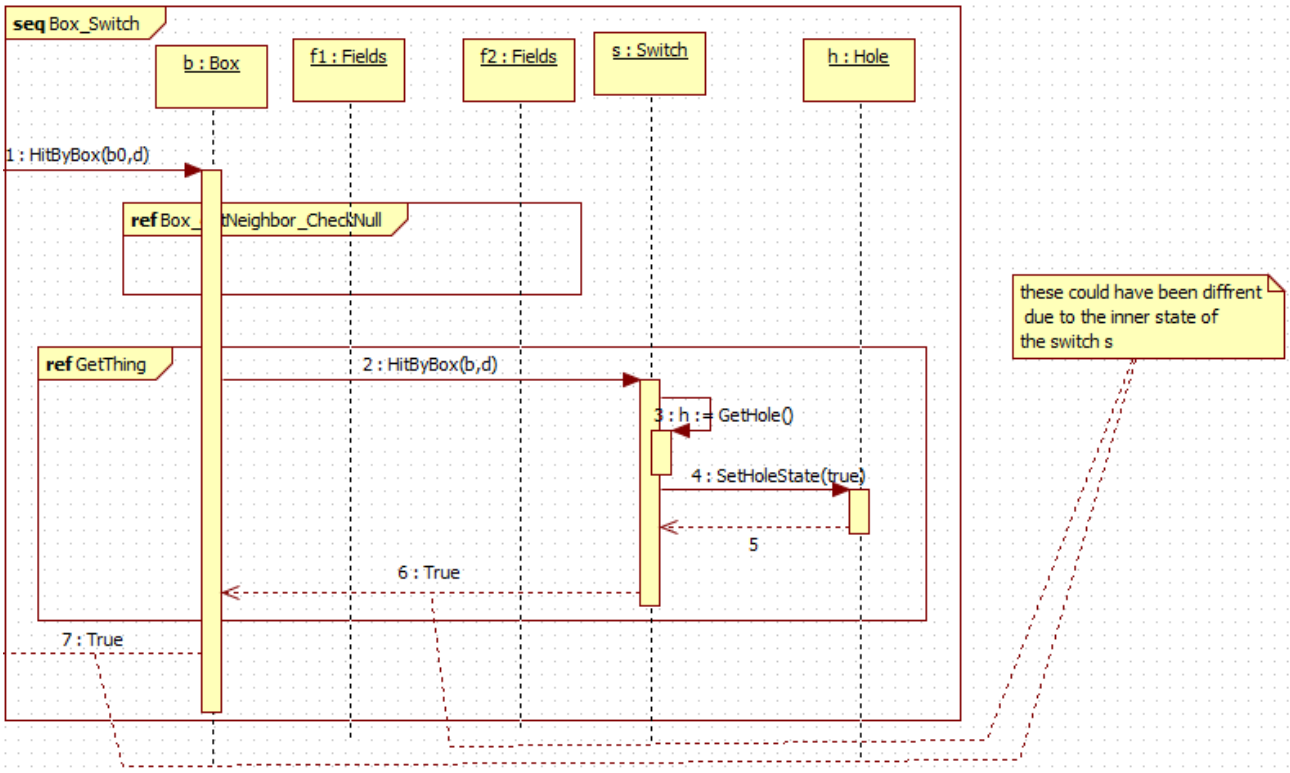
5.4.5 Move Worker Wall: The wall blocks the move



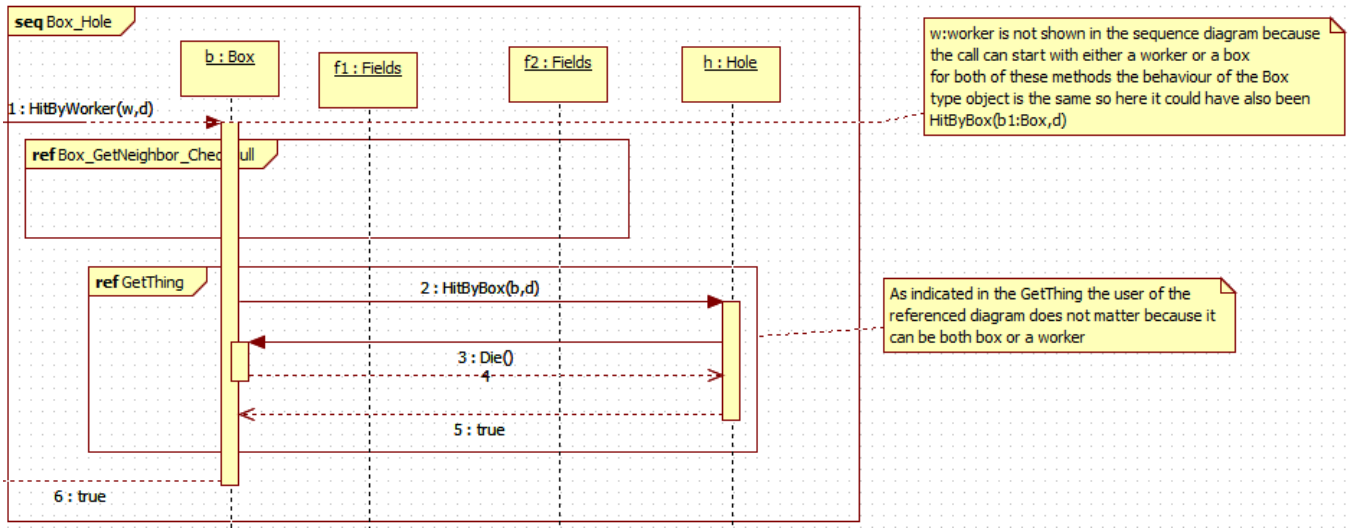
5.4.6 Move Box Worker: The Worker either moves or dies(depends on next field)



5.4.7 Move Box Switch: Switch ON

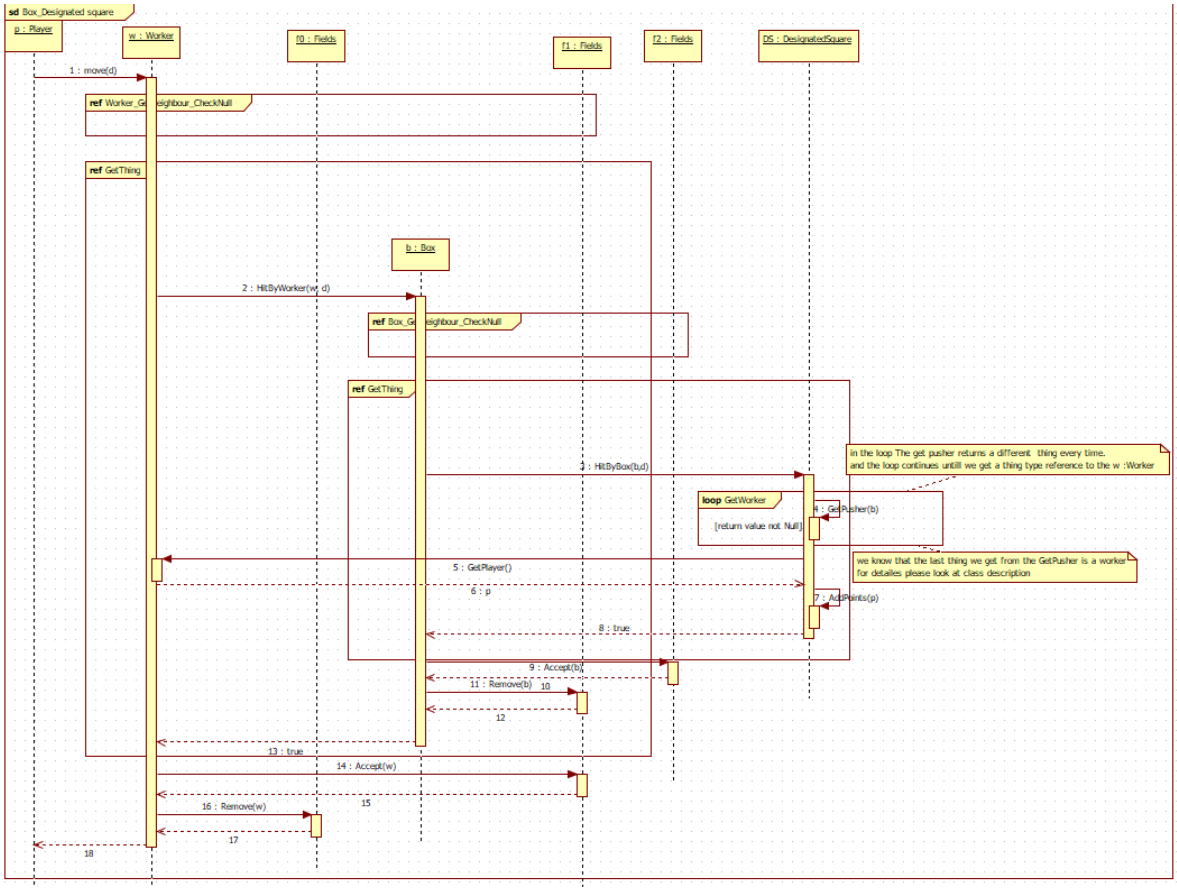


5.4.8 Move Box Hole: Box dies

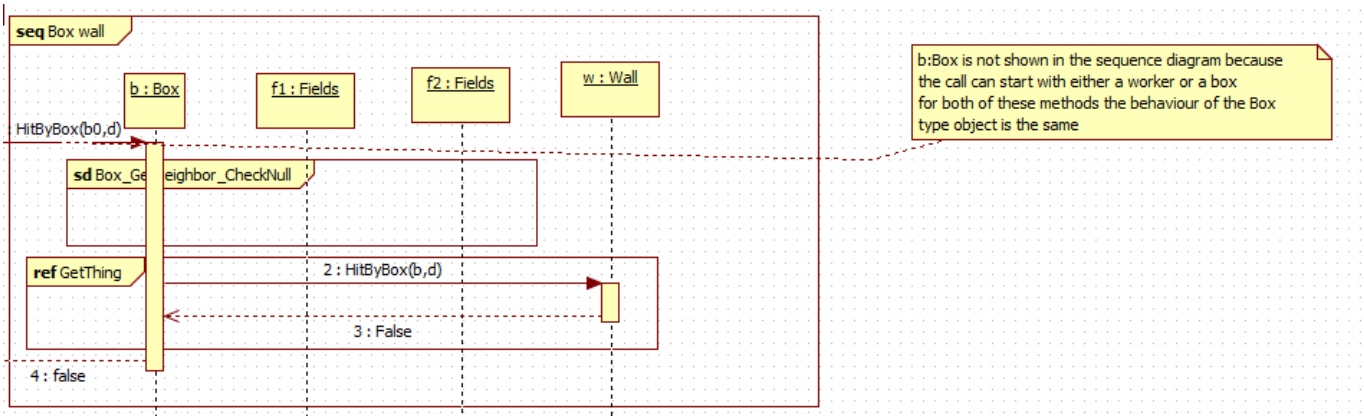


5.4.9 Move Box DesignatedSquare: Player gain points

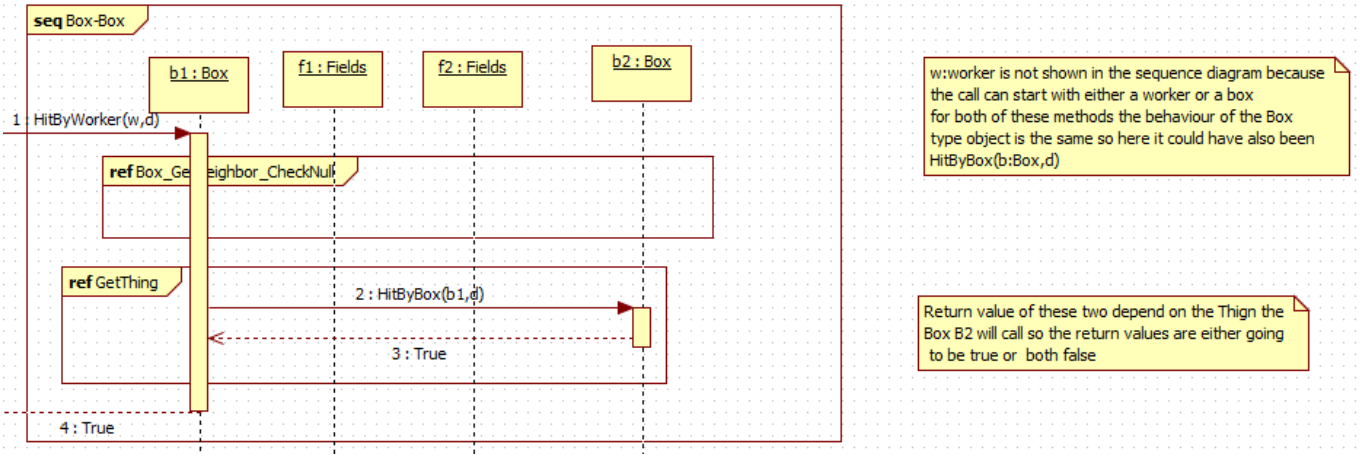
(This sequence Diagram is a reference from the Box DesignatedSquare on the previous document)



5.4.10 Move Box Wall: The Wall blocks the move

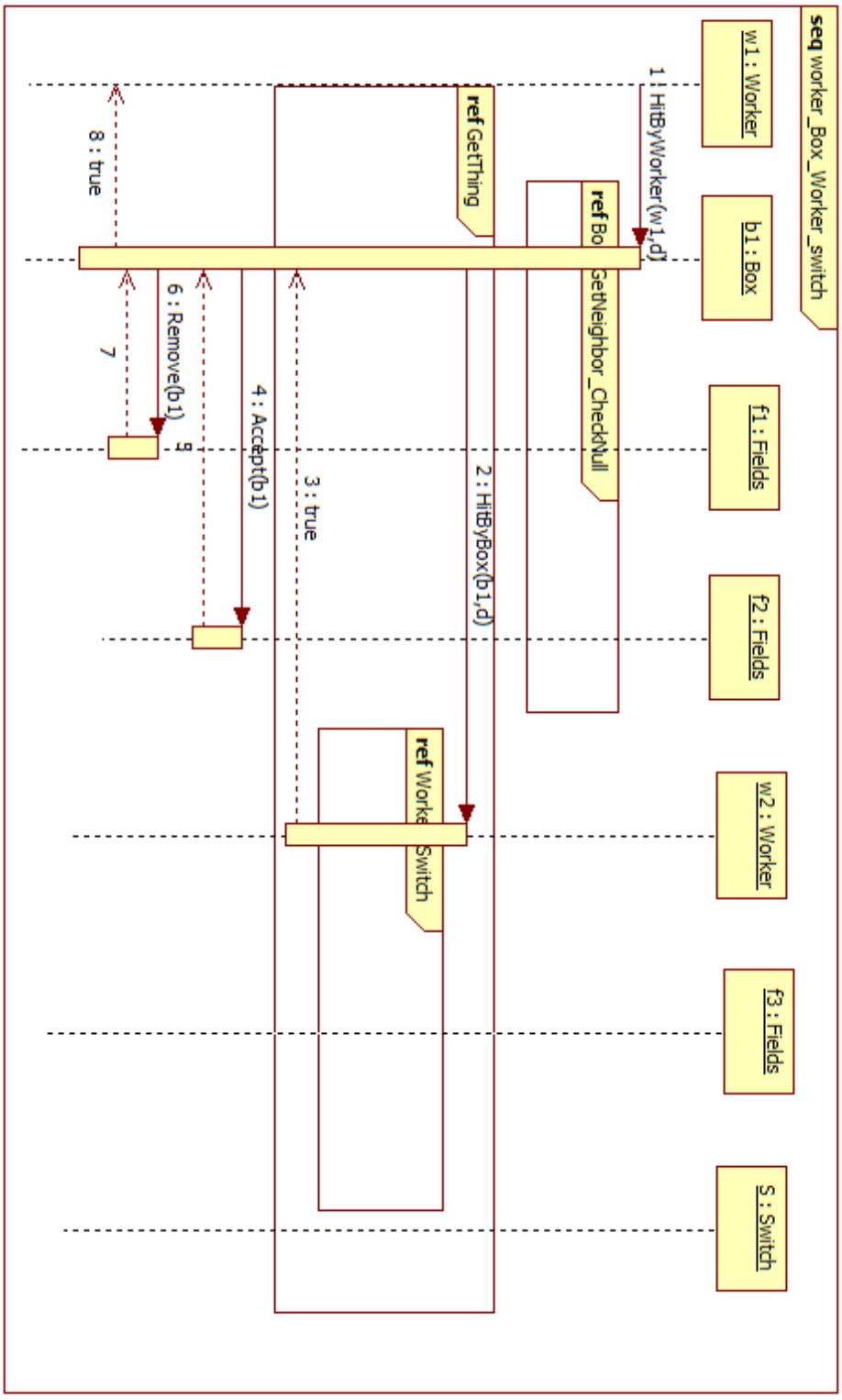


5.4.11 Move Box Box: The two boxes moved to the next field in the direction given

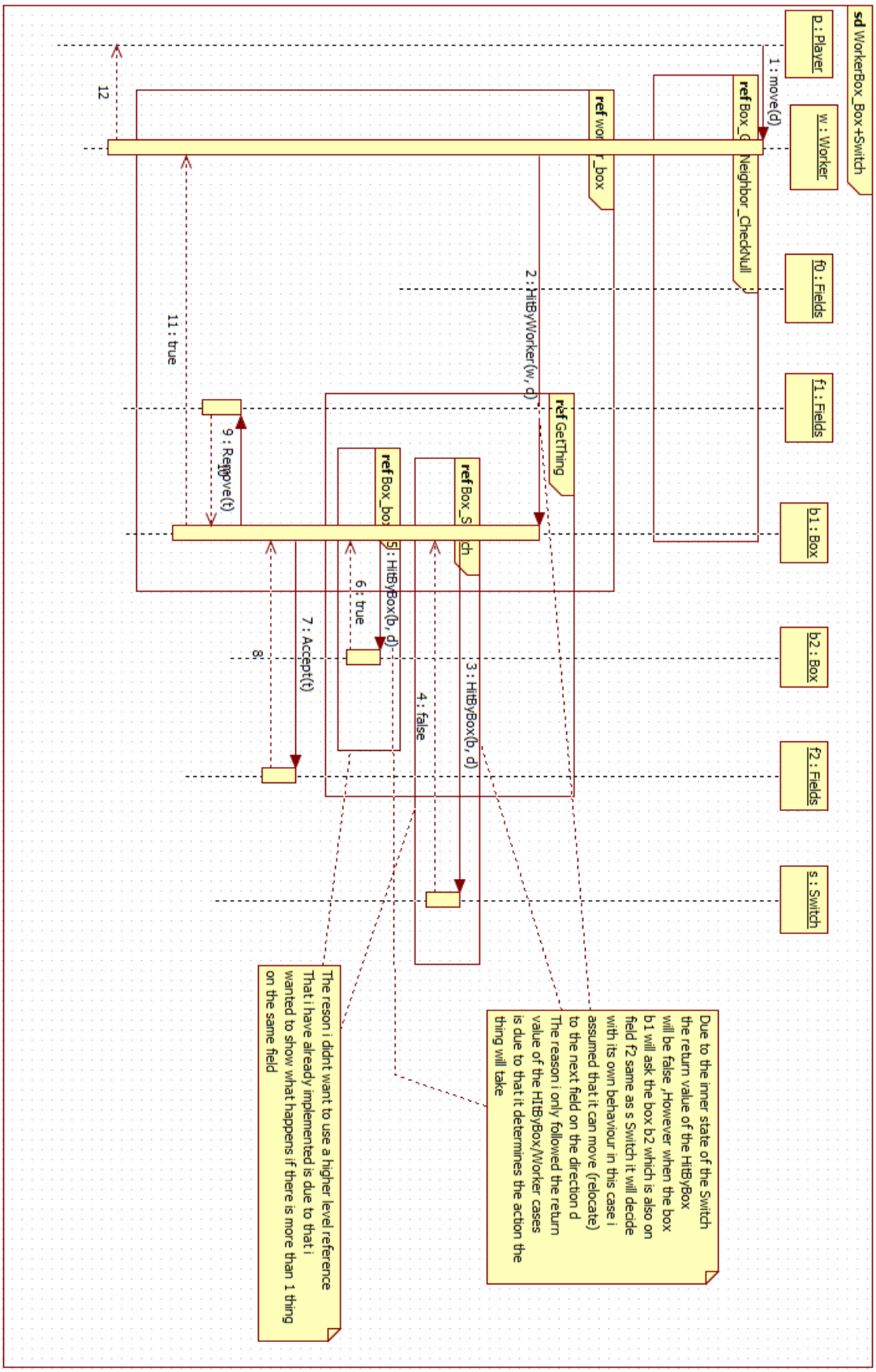


Complex Moves:

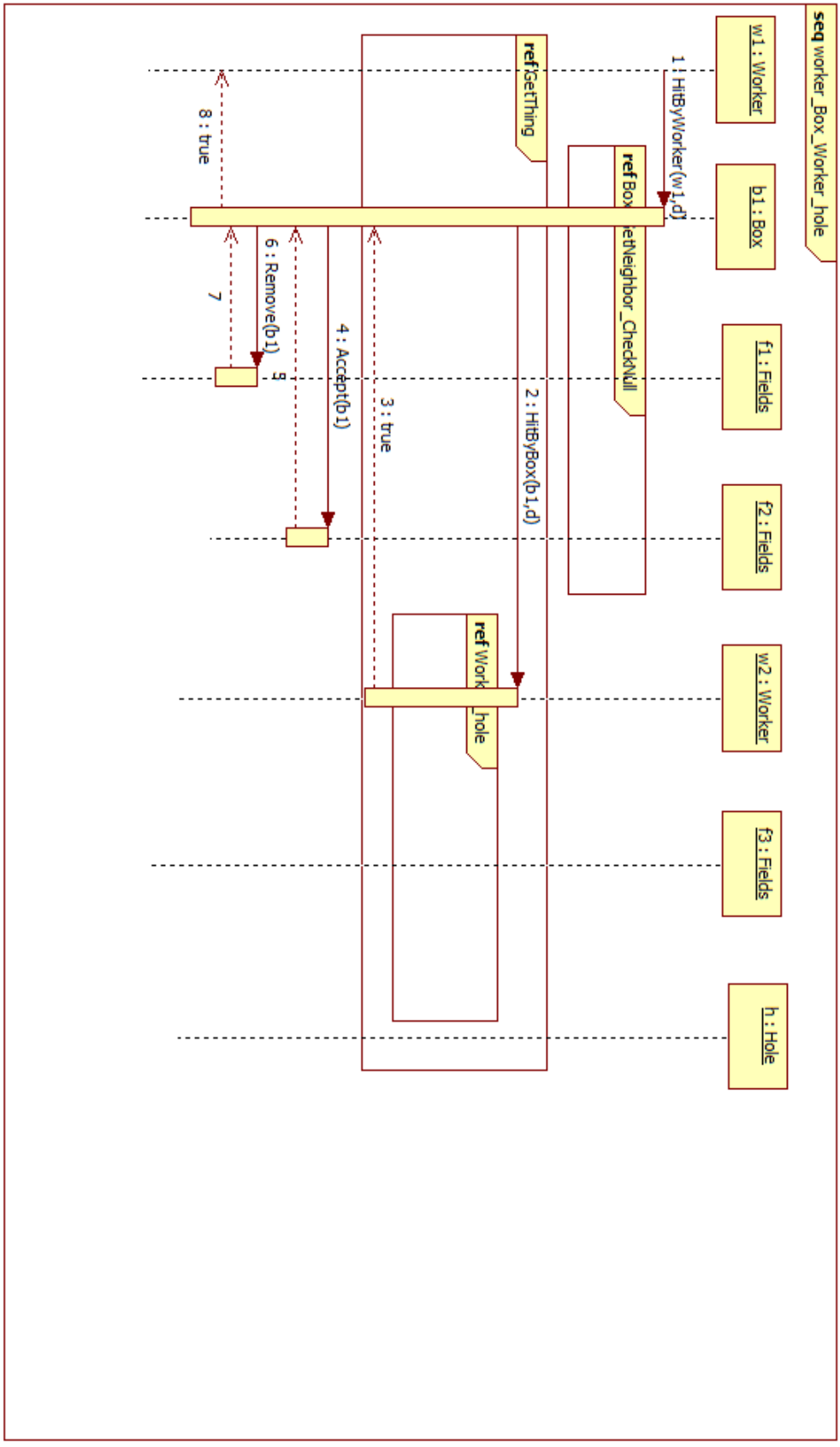
5.4.12 Move Worker Box Worker Switch: Every object moves, the switch state is unchanged (hit by worker)



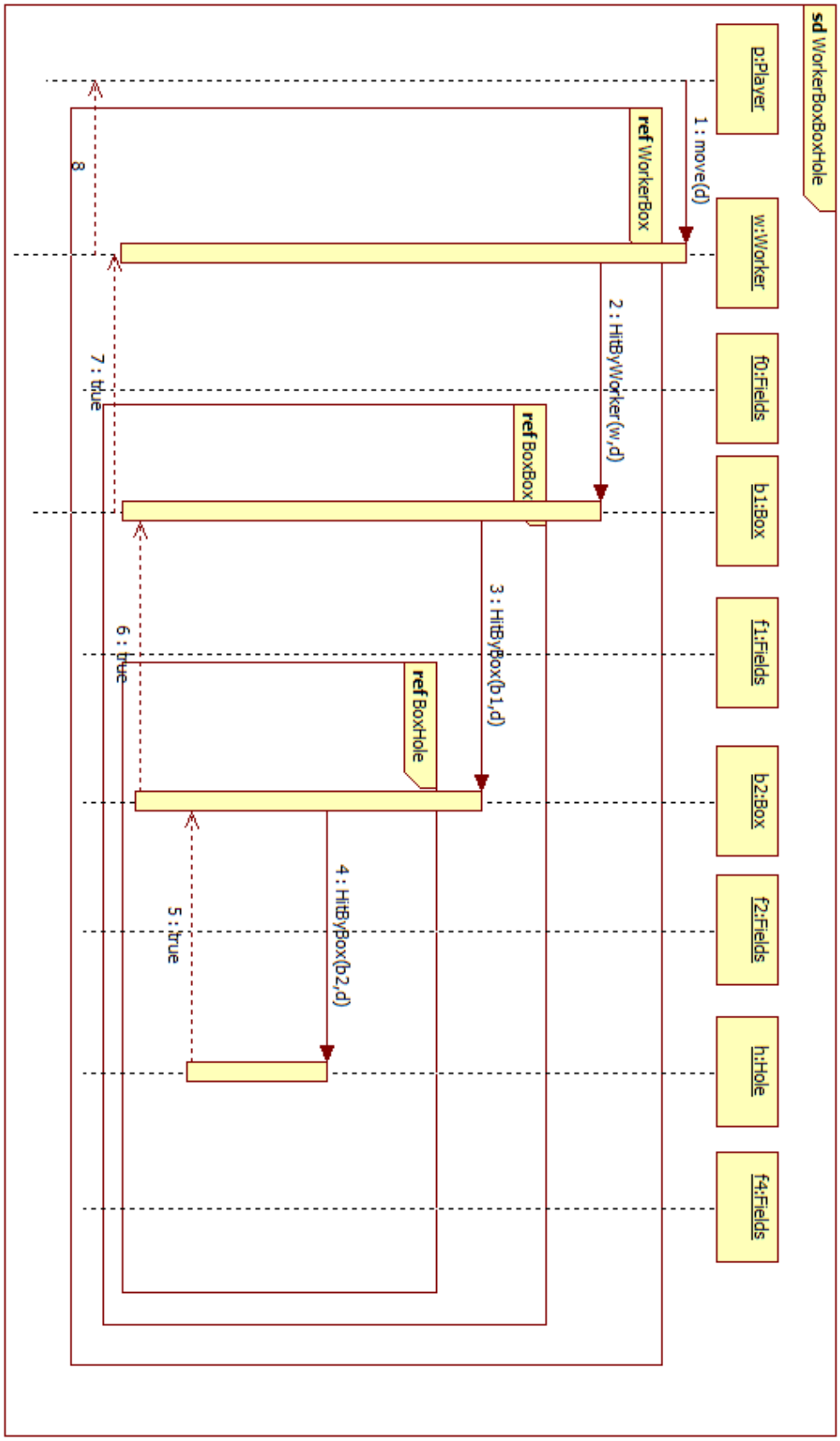
5.4.13 Move Worker Box (Switch+Box): Switch OFF then ON again.



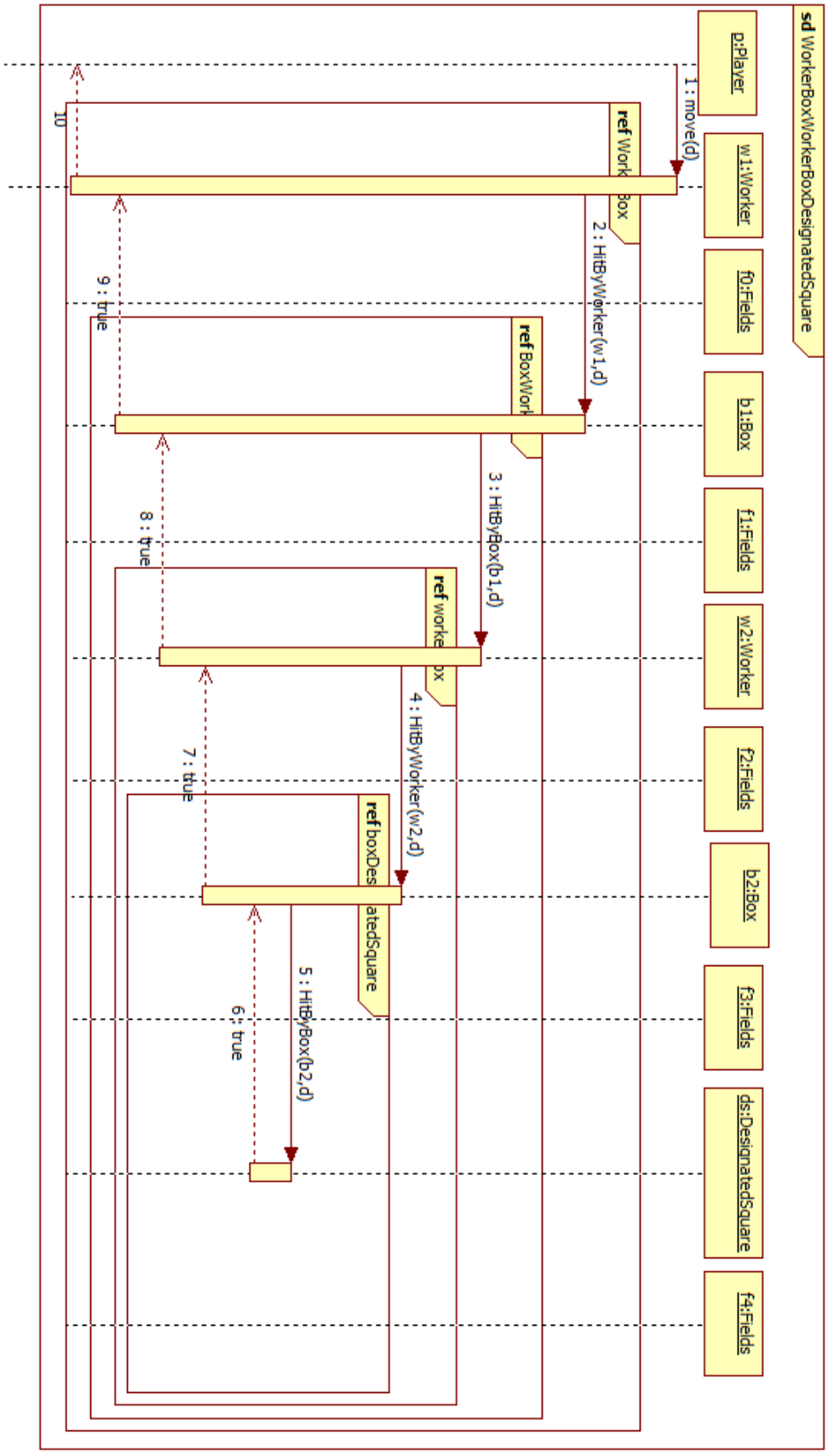
5.4.14 Move Worker Box Worker Hole: The Second worker Dies



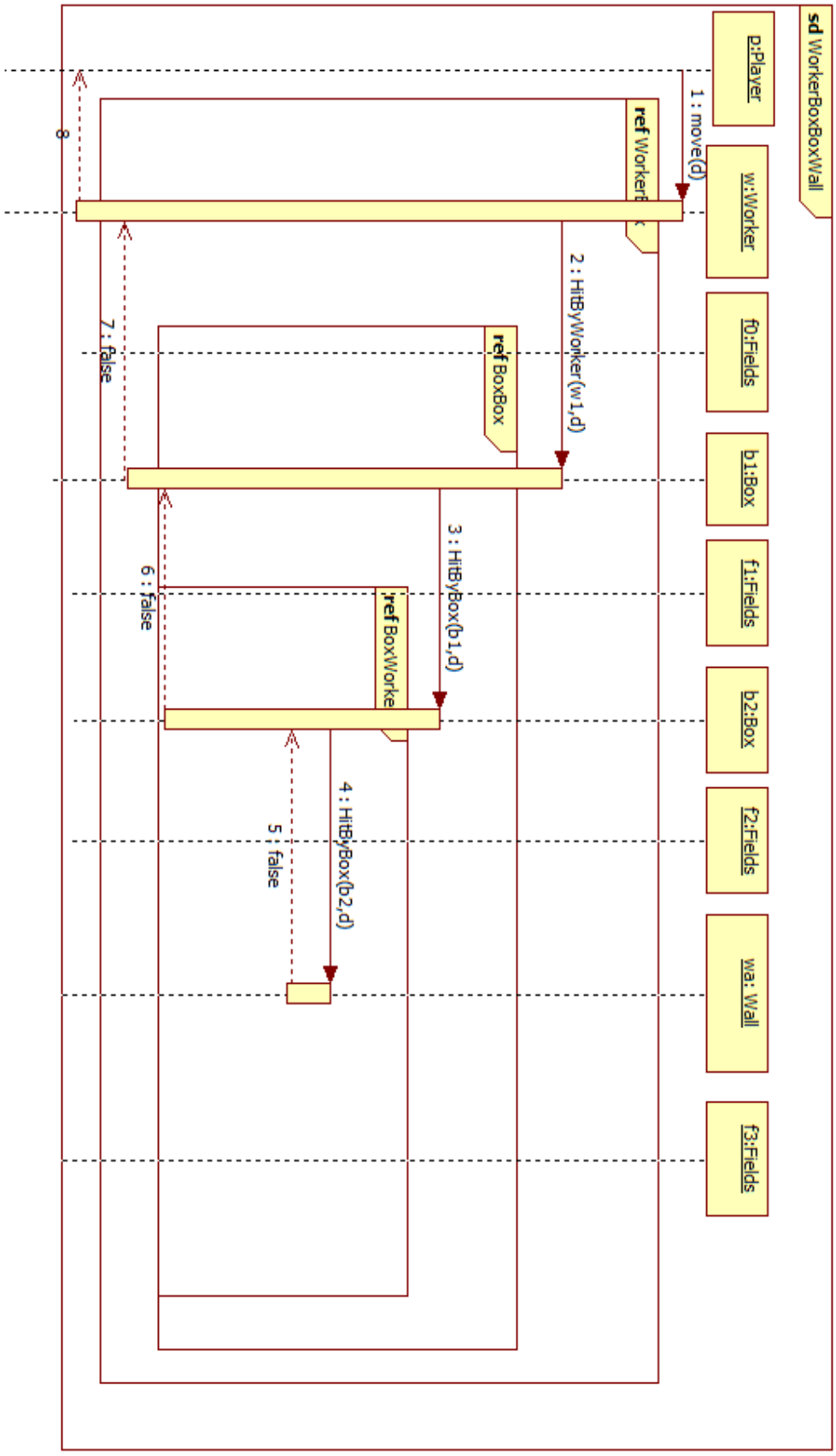
5.4.15 Move Worker Box Box Hole: The second box Dies



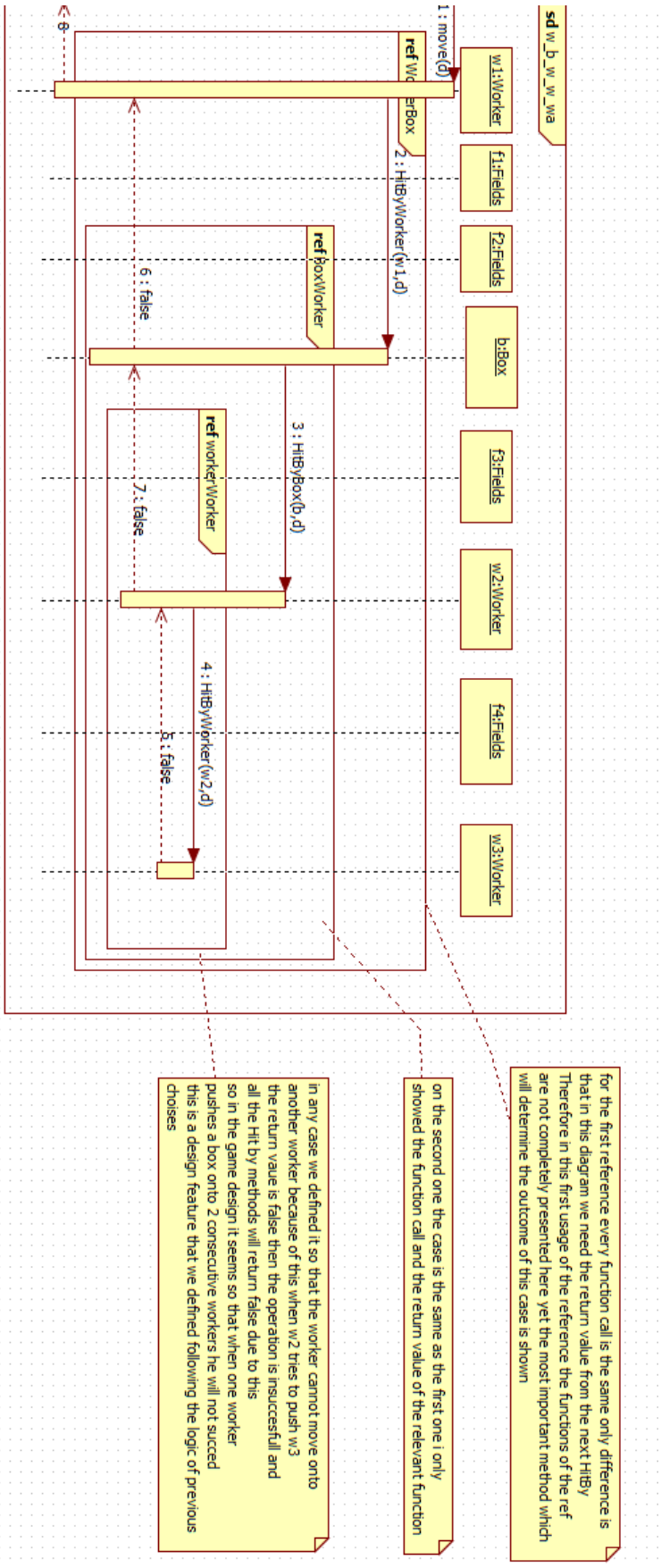
5.4.16 Move Worker Box Worker Box DesignatedSquare: The first Worker who initiated the push gains a Point.



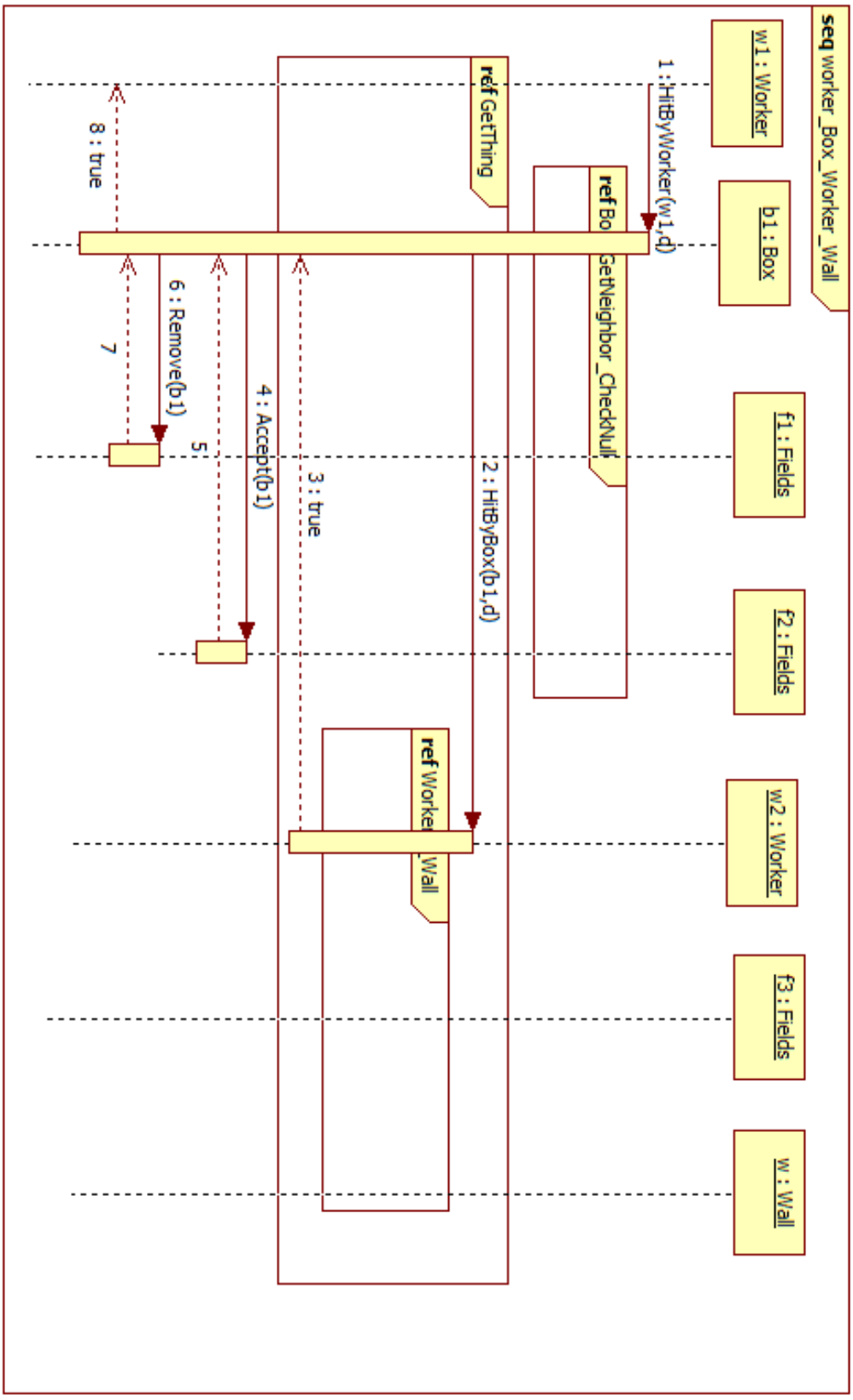
5.4.17 Move Worker Box Box Wall: The Wall blocks the move.



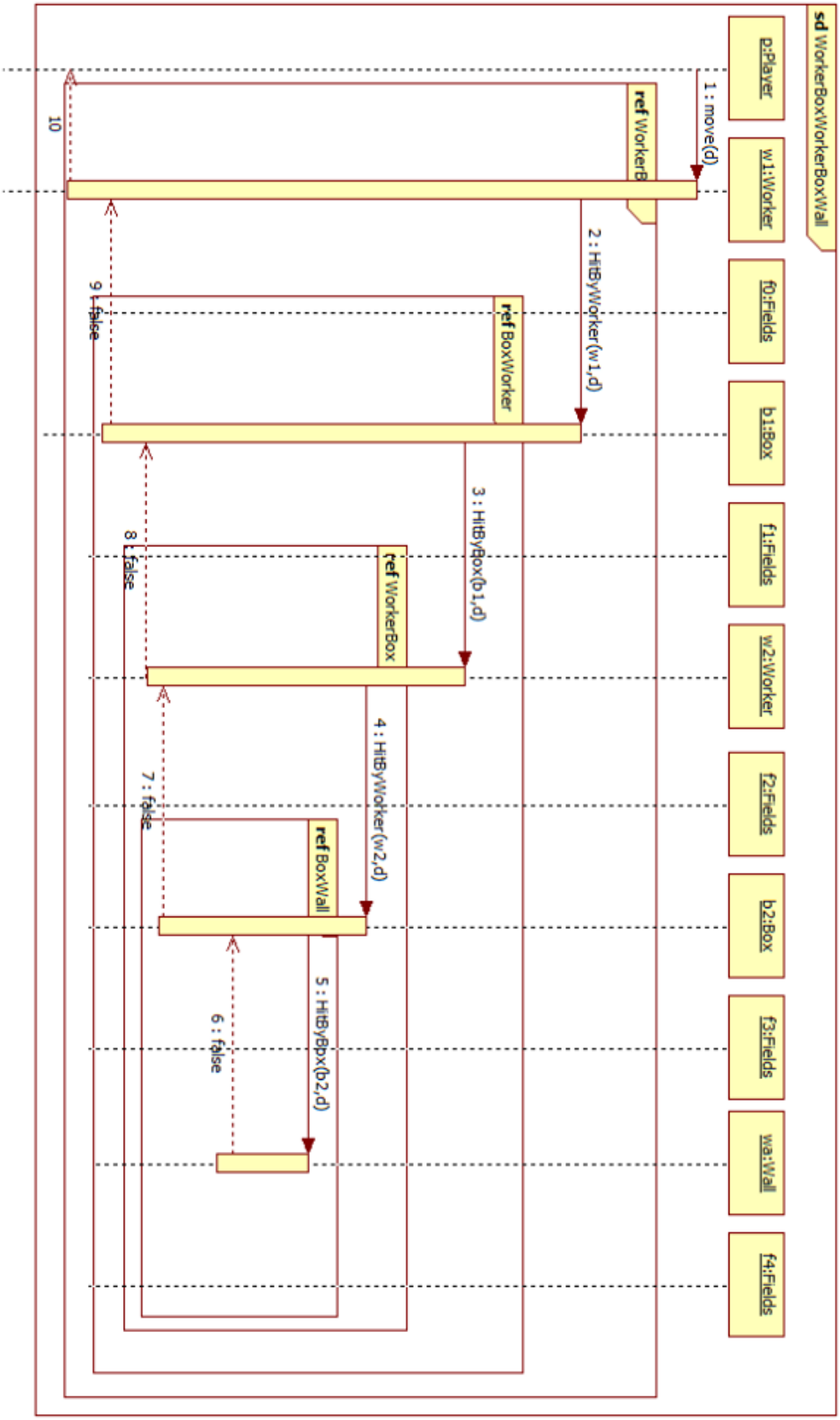
5.4.18 Move Worker Box Worker Worker Wall: The second Worker dies because the third worker is blocked by the Wall.



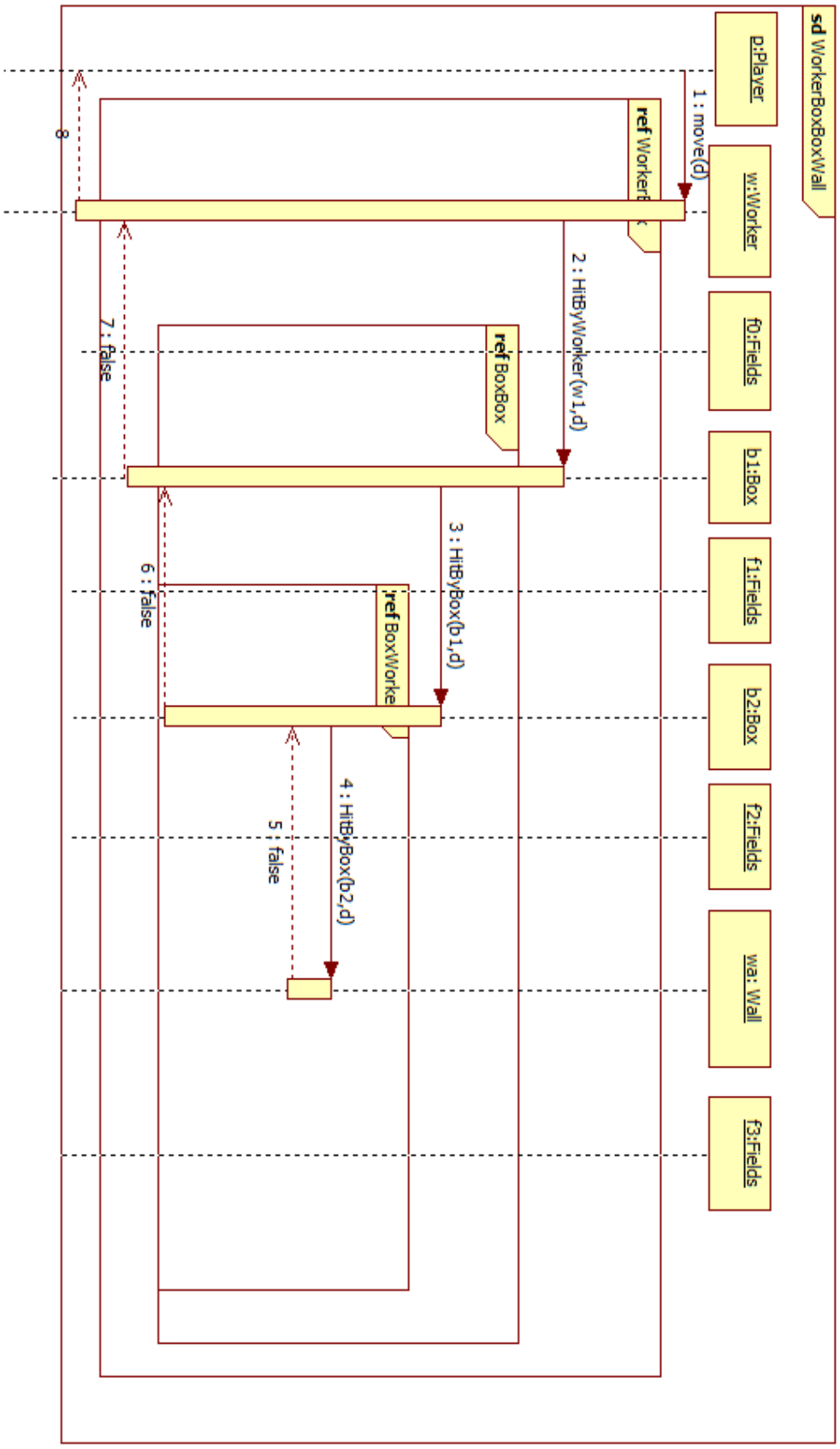
5.4.19 Move Worker Box Worker Wall: The second worker Dies.



5.4.20 Move Worker Box Worker Box Wall: Second worker dies, the second box is blocked by the Wall.



5.4.21 Move Worker Box Box Switch: Second box turns the Switch ON.



5.5 Protocol

Start (date & time)	Duration (hours)	Performer(s) name	Activity description
08.03.2018 14:00	2 hours	Houssam Mehdi, M'ahdi Karray, Khouloud Khezami, Rovshan Shirinli, Tony Azar	[Meeting] Deciding on the use case diagrams and some design improvements are decided.
10.03.2018 13:00	2 hours	M'ahdi Karray, Khouloud Khezami	Use-case , Sequence diagrams
10.03.2018 17:00	1 hour	Houssam Mehdi	New use case diagram
11.03.2018 20:00	4 hours	Houssam Mehdi, M'ahdi Karray, Khouloud Khezami, Rovshan Shirinli, Tony Azar	[Meeting]
12.03.2018 00:00	2 hours	Tony Azar	Use Case Description.
12.03.2018 00:00	4 hours	Rovshan Shirinli, Houssam Mehdi	Collaboration Diagrams, UI, Sequence diagrams, Document.

6. Skeleton program

6.1 Deployment guide

6.1.1 List of files

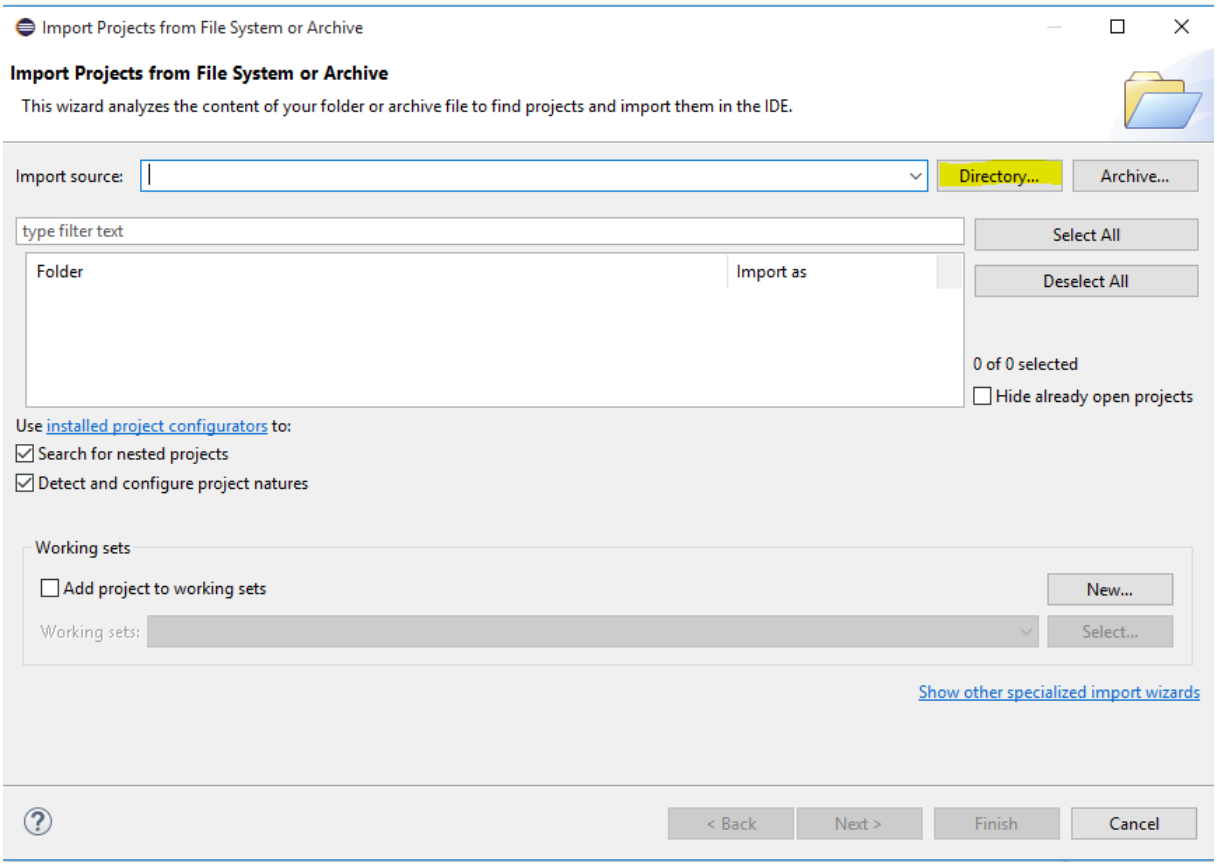
File name	Size	Date	Content
Box.java	4kb	3/19/2018 2:37AM	Box class
Worker.java	4kb	3/19/2018 2:37AM	Worker class
DesignatedSquare.java	3kb	3/19/2018 2:37AM	DS class
Direction.java	1kb	3/19/2018 2:37AM	Enum Directions
Fields.java	3kb	3/19/2018 2:37AM	Fields class
Game.java	9kb	3/19/2018 2:37AM	Game class
Hole.java	1kb	3/19/2018 2:37AM	Hole class
Main.java	18kb	3/19/2018 2:37AM	Main with all the tests
Map.java	2kb	3/19/2018 2:37AM	Map class
Player.java	2kb	3/19/2018 2:37AM	Player class
Switch.java	1kb	3/19/2018 2:37AM	Switch class
Thing.java	1kb	3/19/2018 2:37AM	Thing class
Wall.java	1kb	3/19/2018 2:37AM	Wall class

6.1.2 Compilation

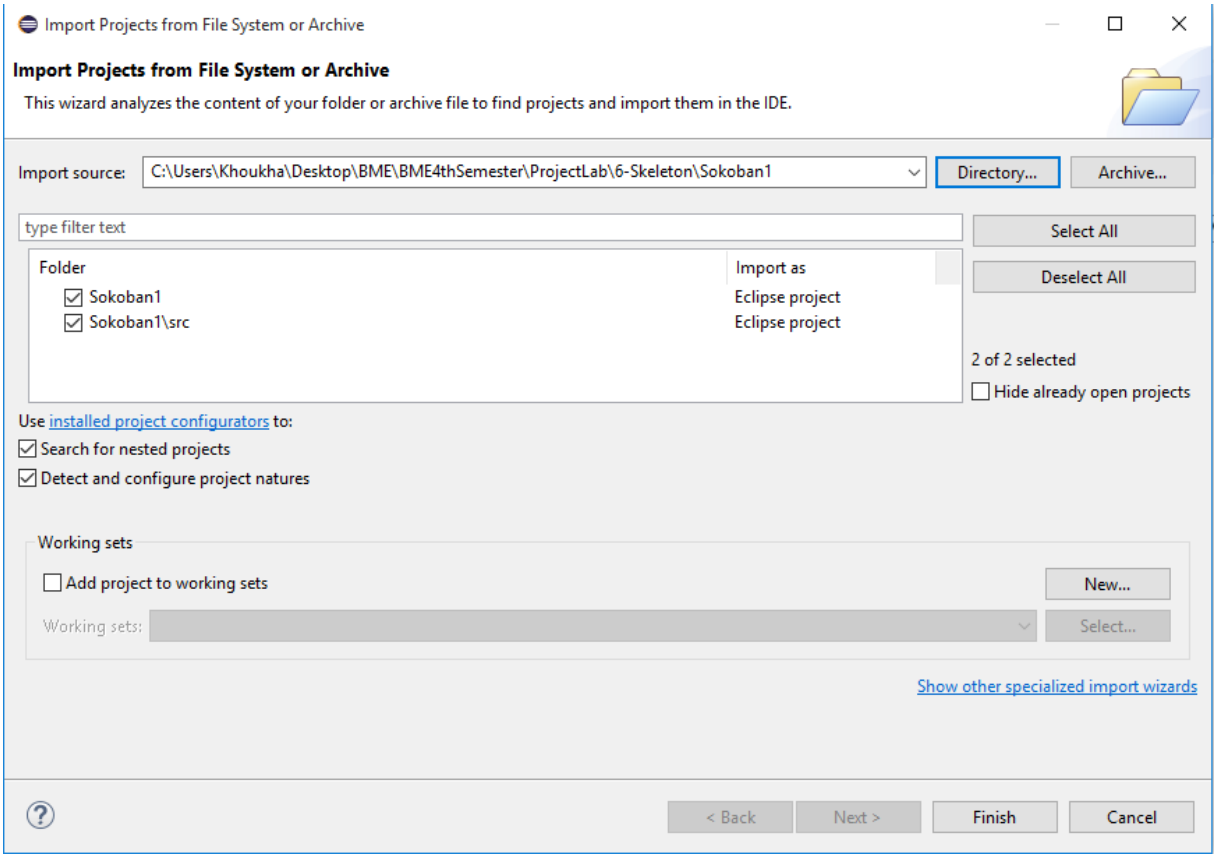
In order to compile the project the Eclipse IDE is needed, for further information about installation you can check the following official website https://www.eclipse.org/downloads/ from where you can download any version compatible with your operating system.
We would recommend installing the latest version of Eclipse IDE.

Now that everything is installed on your computer, download the zip file called Sokoban, and extract it (chlick on the right button of the mouse and you will find extracting options) in any directory of your choice.

After that, launch the Eclipse IDE, go to the menu tab, and in the top left corner you will find File, click on it, then look for Open Projects from File System, the following window will pop up:



Click on Directory and browse your file system to the directory you extracted the zip file in and click on it, and then click on ok.
The importation window should look like this:



Now click on Finish at the bottom and you are ready to go.
The next step is the following, go to the tool bar which should like the following:

And click on the arrow of the highlighted run button and select Main as startup project.

6.1.3 Run

After succesfully applying the steps mentionned in 6.1.2, the Main will run and the Application's console should look like the following:

```
Please enter a command from the following:

 1- Worker Worker
 2- Worker Box
 3- Worker EmptySquare
 4- Worker Switch
 5- Worker DesignatedSquare
 6- Worker Hole
 7- Worker Wall

 8- Box Worker
 9- Box Box
10- Box EmptySquare
11- Box Switch
12- Box DesignatedSquare
13- Box Hole
14- Box Wall

15- Exit
```

Choose a number between 1 and 15 and hit the Enter button of your keyboard.

6.2 Evaluation

Name of the team member	Participation (%)
Rovshan Shirinli	20
Houssam Mehdi	20
Khouloud Khezami	20
M'ahdi Karray	20
Tony Azar	20

6.3 Protocol

Start (date & time)	Duration (hours)	Performer(s) name	Activity description
13.03.2018.	6 hours	Khezami Khouloud	Implemented the classes Player, Worker, DesignatedSquare
15.03.2018	19.03.2018	Khezami Khouloud	Further Implemented the classes Box, Map, Fields, Game and worked on testing and documentation.
15.03.2018 10:00	4 hours	Rovshan Shirinli, M'ahdi Karray	Implemented classes Map,Fields,Direction,Player ,Game.
17.03.2018 18:00	5 hours	Rovshan Shirinli, M'ahdi Karray	Testing functonality of classes Map,Fields,Direction,Player ,Game.

7. Concept of prototype

7.1 Interface definition of Prototype

7.1.1 General description

In this Prototype we have created 21 scenarios that can be played, which actually tests all the functionalities of the game. Whenever the game is started, it will show all the available scenarios that can be played and wait for commands. The scenarios can be chosen with the command „load scenario „ with the number of the scenario after it. After a scenario is loaded, the game will wait for further commands, such as „move worker „ and the direction, which will move a worker to the specified direction, or „save game", „load game" and finally „exit". We have also made the actual tests for each and every scenario, which can also be loaded using the commands „load test" and the number of the test. These tests are saved in a text file; It includes all the commands needed to test a specific scenario, basically it will load a scenario, then „game state", move the worker according to the test direction, and then „game state" again, in order to see how the Things moved on the fields, and if it moved according to the expected output or not.

7.1.2 Input language

Command 1: *load scenario (+number)*

Description: It loads the specified scenario according to the scenario number chosen. The scenarios are shown when you start the game.

Options:

1- Worker Worker (load scenario 1)
2- Worker Box (load scenario 2)
3- Worker EmptySquare (load scenario 3)
4- Worker Switch (load scenario 4)
5- Worker DesignatedSquare (load scenario 5)
6- Worker Hole (load scenario 6)
7- Worker Wall (load scenario 7)

8- Box Worker (load scenario 8)
9- Box Box (load scenario 9)
10- Box EmptySquare (load scenario 10)
11- Box Switch (load scenario 11)
12- Box DesignatedSquare (load scenario 12)
13- Box Hole (load scenario 13)
14- Box Wall (load scenario 14)

Complex Scenarios:

15- Worker Box (Switch+Box) (load scenario 15)
16- Worker Box Worker Box Wall (load scenario 16)
17- Worker Box Box Worker Box DesignatedSquare (load scenario 17)
18- Worker Box Box Worker EmptySquare (load scenario 18)
19- Worker Box Worker (Box+DesignatedSquare) EmptySquare (load scenario 19)
20- Worker Box Worker(Box +DesignatedSqure) Box DesignatedSquare (load scenario 20)

21-Worker Box (worker+DesignatedSquare) Wall (load scenario 21)
22-Worker Box (Switch + Worker) Hole (load scenario 22)
23-Worker Box (DesignatedSquare + Worker) Hole (load scenario 23)
24-Worker Box (Switch+ Box) Hole (load scenario 24)
25- Worker Box (Switch + Box) Worker Hole (load scenario 25)
26- Worker Box Box Worker Box DesignatedSquare Worker Wall (load scenario 26)
27- Worker Box Box Worker Box DesignatedSquare Worker Hole (load scenario 27)
28- Worker Box (Box+DesignatedSquare) Worker Worker (load scenario 28)
29- Worker Box (Box + DesignatedSquare) Worker Box Wall (load scenario 29)

Command 2: *move worker (+direction)*

Description: This command can be called after having loaded a specific scenario, then we're able to move the worker in the direction specified.

Options:

move worker right
move worker left
move worker up
move worker down

Command 3: *save game*

Description: This command can be called after having loaded a specific scenario, then we can save the game using Serialization, all the objects are saved on their specific fields.

Options:
save game

Command 4: *load game*

Description: This command can be called after having loaded the scenario where the game was saved already, then all the Things are placed on their corresponding fields as they were when the game was saved.

Options:
load game

Command 5: *game state*

Description: This command can be called after having loaded a specific scenario, then it will print the state of the game, meaning all the objects and their Fields, plus each Player and his points. (Every map is surrounded by Walls which will not be shown in the „game state" command.

Options:

game state

Command_6: *exit*

Description: If exit is called after loading a scenario, you will quit this scenario and go back to loading scenarios. And if this command is called before loading then it will quit the game.

Options: exit

7.1.3 Output language

Command_1: *load scenario (+number)*

Output: This command will load the specified scenario.
The output will be „Scenario (+number) loaded”

Command_2: *move worker (+direction)*

Output: This command moves the worker in the specified direction.
The output will be „Move executed (+direction)”.

Command_3: *save game*

Output: This command will save the game, by serializing the „game” class, which includes the whole game and put it in the game folder.
When a game is saved, the output is: „Game saved”.

Command_4: *load game*

Output: This command will load a game that has been saved previously from the game folder, and it will make the current game as the loaded game. All the saved Things will be loaded on their corresponding fields, along with the Players and their points.
When a game is loaded, the output is: „Game loaded”.

Command_5: *game state*

Output: This command will check the Map, and print all the Things on it with their fields, players with their points etc. The function will go through every field (ignoring the border fields since they have walls) and get the things on each field.
Example of output:

```
game state
Game size 10
class Worker [1][1]
class Box [1][2]
class Box [1][3]
class Worker [1][4]
```

```
class Box [1][5]
class DesignatedSquare [1][6]
Player 0 points: 0
Player 1 points: 0
```

Command 6: *exit*

Output: If exit is written before loading a scenario, the game will terminate. If it is written after loading a scenario the scenario will exit and go back to loading scenarios.

7.2 Real use-cases

Use-case name	Move Worker Wall
Short textual description	Worker tries to move and there is a Wall
Actors	Player
Dialog, scenario	The wall stops the move of the Worker

Use-case name	Move Worker Hole
Short textual description	Worker moves to a Hole
Actors	Player
Dialog, scenario	When Worker steps on Hole the Worker dies

Use-case name	Move Worker Worker
Short textual description	A Worker pushes a Worker
Actors	Player
Dialog, scenario	A Worker cannot push another Worker directly

Use-case name	Move Worker Switch
Short textual description	a Worker tries to move and there is a Switch
Actors	Player
Dialog, scenario	The switch does not change it's state when the worker steps on it.

Use-case name	Move Worker Box
Short textual description	Worker moves the Box to an empty field
Actors	Player
Dialog, scenario	The Box is hit by the worker

Use-case name	Move Worker DesignateSquare
Short textual description	Worker moves into a DS
Actors	Player
Dialog, scenario	When the worker moves into the DS, nothing happens.

Use-case name	Move Box Hole
Short textual description	When a Box is moved on a Hole
Actors	Player
Dialog, scenario	When the Box steps on a Hole, it dies.

Use-case name	Move Box Switch
Short textual description	A Worker pushes a Box or Boxes onto the Switch the inner state of the corresponding Switch and the associated Hole will be changed.
Actors	Player
Dialog, scenario	When the Box is on the Switch it turns the Switch ON

Use-case name	Move Box Wall
Short textual description	When a Box is moved on a Wall
Actors	Player
Dialog, scenario	The box doesn't move because the wall blocks it.

Use-case name	Move Box Worker
Short textual description	When a Box is moved on a Worker
Actors	Player
Dialog, scenario	The Box pushes the Worker, what happens depends on the next field.

Use-case name	Move Box Box
Short textual description	When a Box is moved on a Box
Actors	Player
Dialog, scenario	The Box is pushed, if it's an empty field, both of them move to the next field.

Use-case name	Move Box DesignatedSquare
Short textual description	Box is moved on a DS
Actors	Player
Dialog, scenario	When the Box steps on the DS, the Worker who initiated the move gets a point.

Use-case name	Move Worker Box (Switch + Box)
Short textual description	A Worker pushes a Box on another Box which is on a Switch
Actors	Player
Dialog, scenario	The status of the switch is ON since it has a box, when the box is pushed out by the first box, it goes OFF and then the first box steps on the switch and turns it ON again.

Use-case name	Move Worker Box Worker Box Wall
Short textual description	The Worker pushes the Box that pushes the second Worker on a Box that is pushed on a Wall
Actors	Player
Dialog, scenario	When a Worker pushes a Box on a Worker then the worker will try to push the Box in the same direction if unsuccessful it will be crushed by the box trying to hit it. It will die.

Use-case name	Move Worker Box Worker (Box + DesignatedSquare (DS)) EmptyField

Short textual description	A worker pushes a box onto another worker which will push the Box out of the field that the DS is on.
Actors	Player
Dialog, scenario	When the second box is pushed out, the worker that initiated the move will lose points.
Use-case name	Move Worker Box Box Worker EmptySquare
Short textual description	The Worker pushes the Box which pushes the second Box on a Worker then an EmptySquare.
Actors	Player
Dialog, scenario	When a Worker pushes a Box on a Box it checks the next field where it finds a worker, so every object moves on the next field in the given direction

Use-case name	Move Worker Box Box Worker Box DesignatedSquare
Short textual description	The Worker pushes two boxes, which pushes a worker then another box into a DS.
Actors	Player
Dialog, scenario	When the Worker pushes the two boxes that pushes the second Worker that pushes the third Box on the DesignatedSquare the first Worker is the one that gets the point and every object moves on the next field on the given direction.

Use-case name	Worker Box (worker+DesignatedSquare) Wall
Short textual description	Worker pushes a box which pushes a worker, that is placed on a designatedSquare, to a Wall
Actors	Player
Dialog, scenario	When the Worker pushes the box to the designated square he gets added points, however this move will push the second worker to the wall, and the second worker will die.

Use-case name	Worker Box (Switch + Worker) Hole
Short textual description	Worker pushes a box which pushes a worker, that is placed on a switch, towards a Hole
Actors	Player
Dialog, scenario	When the Worker pushes a box to the next field that has a worker and a switch, the state of the switch is on, and the worker gets pushed to the hole whose state is off so he doesn't die

Use-case name	Worker Box (DesignatedSquare + Worker) Hole
Short textual description	Worker pushes a box to a designatedSquare that has a Worker on it which gets pushed to a Hole
Actors	Player
Dialog, scenario	Worker pushes a box to a field that has a worker and a designatedSquare, the worker's player that initiated the move gets additional points, and the second worker gets pushed to a hole whose state is on, the second worker dies

Use-case name	Worker Box (Switch+ Box) Hole
Short textual description	Worker pushes a box that pushes a box placed on switch toward a hole
Actors	Player
Dialog, scenario	Worker pushes Box to a field that has a Switch and a Box, the second Box gets pushed to a hole whose state is on, the second Box disappears

Use-case name	Worker Box (Switch + Box) Worker Hole
Short textual description	Worker pushes a box which pushes a box that is placed on a switch, which pushed a worker to a hole
Actors	Player
Dialog, scenario	Worker pushes a box which pushes a box that is placed on switch, the switch state remains on, the second box pushes the second to a hole whose state is on, so the second worker dies

Use-case name	Worker Box Box Worker Box (DesignatedSquare +Worker)Wall
Short textual description	Worker pushes two boxes that push a worker that pushes a box to a designatedSquare that has a worker that is pushed to a wall
Actors	Player
Dialog, scenario	Worker pushes two boxes that push a worker that pushes a box to a designatedSquare that has a worker that is pushed to a wall, the second player dies, and the first worker's player gets additional points

Use-case name	Worker Box Box Worker Box (DesignatedSquare +Worker)Hole
Short textual description	Worker pushes two boxes that push a worker that pushes a box to a designatedSquare that has another worker on it that gets pushed to a hole
Actors	Player
Dialog, scenario	The first Worker pushed two boxes that push a worker that pushes a box to a designated square, the first worker's player gets additional points, however the third worker that was placed on a designatedSquare gets pushed to a hole whose state is on so he dies

Use-case name	Worker Box Box (DesignatedSquare+Worker) Worker
Short textual description	Worker pushes two boxes that push a worker that is placed on a designatedSquare, the second worker will be pushed to a third worker
Actors	Player
Dialog, scenario	Worker pushes two boxes that push a worker that is placed on a designatedSquare, the second worker will be pushed to a third worker, this move will not occur as the second worker can not push the third worker

Use-case name	Worker Box (Box + DesignatedSquare) Worker Box Wall
Short textual description	Worker pushes a box that pushes a box placed on a DS the second box pushes a worker that pushes a box towards a wall
Actors	Player
Dialog, scenario	Worker pushes a box that pushes a box placed on a DS the second box pushes a worker that pushes a box towards a wall, this move will not occur as the box can not push the wall

Use-case name	Worker Box Worker(Box +DesignatedSqure) Box DesignatedSquare
Short textual description	Worker pushes a box that pushes a worker that pushes a box placed on a designated square, the second box pushed another box towards a designated square
Actors	Player
Dialog, scenario	Worker pushes a box that pushes a worker that pushes a box placed on a designated square, the second box pushed another box towards a designated square, the first worker's player's point will be subtracted as he moved the second box from the DS, and the first worker's player will get additional points because the third box got pushed to a DS

7.3 Test plan

Name of the test-case	1- Worker Worker
Goal	To show the interaction between 2 workers. Worker 1 is able to move using „move worker" commands and interact with nearby objects.
Short description	Workers are added in indexes [1][1] and [1][2] because as defined before all games has walls on the edges of the map. The first worker will try to go onto the second worker

Name of the test-case	2- Worker Box
Goal	To see if a worker can move a box and if the box can get stuck in a corner
Short description	We try to push the box until the wall on the right side and go down and push in up direction.

Name of the test-case	3-Worker EmptySquare
Goal	To see if a worker can step on an empty square, this is defined as a basic move so we check.

Short description	The worker will move along the walls while stepping on different empty squares that are adjacent to the walls.

Name of the test-case	4-Worker Switch
Goal	To move the worker onto a switch and then onto the adjacent hole where the worker will die.
Short description	We will move the worker right twice when he first moves he will be on the same field as the switch then he will disappear(die) from the map on the second move.

Name of the test-case	5-Worker DesignatedSquare
Goal	Worker will step on a Ds
Short description	We will move the worker right and make him step on the ds then leave the ds. Thus observing the basic interactions between 2 types.

Name of the test-case	6-Worker Hole
Goal	To see if a worker will die when stepping on the Hole
Short description	First we will move the worker down and move right ,then go up to step on the hole. It should result in death of the worker.

Name of the test-case	7-Worker Wall
Goal	Worker will attempt to push the walls
Short description	First he will try to go up as defined before the walls will not allow the worker to relocate so he won't change positions then he will try to go in another direction.

Name of the test-case	8-Box Worker
Goal	Try to push a box onto another worker, we will push until the second worker is between the box and the wall.
Short description	We will keep moving the first worker right. As a result we will kill the second worker.

Name of the test-case	9- Box Box
Goal	Testing if two adjacent boxes are moveable.
Short description	We will first try to move the worker to right which shall result in no change, then we will move the worker to the

	opposite side of the boxes and try to push there we should expect the same result.

Name of the test-case	10- Box EmptySquare
Goal	Whether a box can be moved to an EmptySquare
Short description	We will try to push the box around the map until we push it into a corner where it will get stuck.

Name of the test-case	11- Box Switch
Goal	We will push the box onto a switch which will turn on the hole that the switch is connected to.
Short description	We will move the worker onto the hole first which will result in nothing after we push the box onto switch the same operation will result with death of the worker.

Name of the test-case	12- Box DesignatedSquare
Goal	We will push the box onto DS and remove it from there while observing the points of players.
Short description	When the box is moved onto the DS it will add points to the player, then the player will move around, and the points will not change. Later when the player pushes the box out of the DS, he will lose points.

Name of the test-case	13 -Box Hole
Goal	We will observe whether the box disappears when pushed onto a hole.
Short description	We will move right twice which will first push the box onto a hole then the worker will die.

Name of the test-case	14-Box Wall
Goal	Testing simple interaction between things
Short description	We will push the box on the wall which shall result in no change.

Name of the test-case	15- Worker Box (Switch+Box)
Goal	Try to observe what happens when the box on a switch changes
Short description	The box will be pushed by the worker onto the field that has the Switch+Box

Name of the test-case	Worker Box Worker Box Wall
Goal	Observing the squeeze case when a worker is pushed by a box onto another immoveable box
Short description	We will move the worker right and second worker will die and the box will go to the place of the second worker.

Name of the test-case	Worker Box Box Worker Box DesignatedSquare
Goal	To see if 2 adjacent boxes are still immovable in the complex situations
Short description	When we try to move the first worker in direction right it will result in no change.

Name of the test-case	Worker Box Box Worker EmptySquare
Goal	To see if 2 adjacent boxes are still immovable in the complex situations
Short description	Empty square is shown to indicate that there is no wall after the second worker, when trying to push 2 boxes in a direction that puts them on the same push line will result in no change but when we separate them by moving them from up or down we will be able to move both boxes afterwards.

Name of the test-case	Worker Box Worker (Box+DesignatedSquare) EmptySquare
Goal	To test that the points is added to the correct player
Short description	When the first worker moves right the second worker will push the box out of the ds and lose points then when the first worker pushes the box onto to the ds while shifting the second worker out of the ds will gain points.

7.4 Support programs for testing

It's undefined for the moment.

7.5 Protocol

Start (date & time)	**Duration (hours)**	**Performer(s) name**	**Activity description**
28/03/2018 14:00	1 hour	All team members	Meeting
28/03/2018 15:00	6 hours	All team members	Documentation

8. Detailed plans

8.1 Design level plan of classes

8.1.1 Worker

- **Responsibility**

Represents a Worker on a field. When the game starts, the map generates the same number of worker and player, who have a bijective relationship, a worker belongs to one player, and a player belongs to one worker. The Worker can be moved by either its corresponding player, or by boxes. The main objective of the Worker is to push boxes to designated squares, when this is done, the corresponding player gets points, however if a worker pushes a box away from a designated square, its corresponding player gets his points subtracted.

- **Superclasses**

Thing

- **Interfaces**

Serializable

- **Attributes**

Field: this attribute is protected and inherited from the super class, this attribute stores an object of type Fields, its main role is indicating where the Worker is stored in the map.
The field attribute is set to the initial spot of the worker when the worker is created.
This attribute will get updated whenever a worker is pusher or moved.

Previous: this attribute is public and inherited from the super class, this attribute stores an object of type Thing, its main role is indicating what made the Worker move, which means if the worker was pushed by box, then the previous attribute will be set to that particular box, however if the worker is moved by a player, the previous attribute will be set to null.
This is needed to handle for instance the following case and others: worker box worker box designated square, the points should be added to the first worker, therefore tracing back should be made.

Player: this attribute is private, this attribute stores an object of type Player, its main role is indicating which player the worker belongs to.

- **Methods**

Public Worker(Player p, Fields f): this is the constructor of the Worker class which takes two parameters, a player ’p’ and a field ’f’, the player attribute will be set to the passed parameter p and the attribute field will be set to the passed parameter p, the previous attribute will be set to null, as initially no object is pushed.

Public move(Direction d): this method takes a direction as a parameter, this method will mainly be called by the corresponding player of the worker.
This method will set the previous attribute to null, then will get the current field and the neighbor of the current field based on the direction ’d’ passed in arguments and we get the Things on it. If there no Thing on the field then the move is made directly and the worker is

accepted in the neighboring field and removed from the current field as well as the field attribute will be updated. However, if there is things on the neighboring field, then based on its state, the move is decided if it is free, then the move is made and the worker is accepted in the neighboring field and removed from the current field as well as the field attribute will be update, if the neighboring field is not free, then we call the hitByWorker(this) of each thing on the neighboring field, if at least one returns false, the move is not made.

Public void die(): this method is called when the Worker is pushed into a wall or into a hole, this method will remove the worker from the current field and set the player into null.

Public Player getPlayer(): this method return the player of the worker

Public boolean HitByBox(Box b, Direction d): this method is called by a box to push the worker, this method will get the neighboring field of the current field based on the direction 'd' passed in arguments and we get the Things on it. If there no Thing on the field then the move is made directly and the worker is accepted in the neighboring field and removed from the current field as well as the field attribute will be updated, the previous attribute will be set to Box 'b' passed in the argument, and the method return true. However, if there is things on the neighboring field, then based on its state, the move is decided if it is free, then the move is made and the worker is accepted in the neighboring field and removed from the current field as well as the field attribute will be updated, the previous attribute will be set to Box 'b' passed in the argument and the method return true, if the neighboring field is not free, then we call the hitByWorker(this) of each thing on the neighboring field, if at least one returns false, the move is not made and this method return false.

Public boolean HitByWorker(Worker w, Direction d): this method retuns false

8.1.2 DesignatedSquare

- **Responsibility**

It represents a designated square on a field. If a Box is pushed onto it, the player of the worker that initiated the move gets points added to his score. If a Box is pushed from it, the player of the worker that initiated the move gets points subtracted from his score.

- **Superclasses**

Thing

- **Interfaces**

Serializable

- **Attributes**

Field: this attribute is protected and inherited from the super class, this attribute stores an object of type Fields, its main role is indicating where the DesignatedSquare is stored in the map.
The field attribute is set when the designatedSquare object is created.

Free: this attribute is private and of type **boolean**, this attribute indicates if the designated square has a box on it or not.

- **Methods**

Public DesignatedSquare(Fields f): this is the constructor of the designated square, it sets the field attribute to the ’f’ passed as a parameter and sets the free to true.

Public boolean HitByWorker(Worker w, Direction d): this method will check the value of the free attribute, if it is true, the worker ’w’ will be accepted the field on which the designated square is located and removed from its current field and the method return true, however, the player of the worker who initiated the move will have points subtracted from his score, the worker ’w’ will be accepted the field on which the designated square is located and removed from its current field and the method return true.

Public boolean HitByBox(Box b, Direction d): this method will check the value of the free attribute, if it is true, the Box ’b’ will be accepted the field on which the designated square is located and removed from its current field, the player of the worker that initiated the move will have points added to his score and the method return true, however nothing happens to the score of the player but the rest is the same as if free is true.

Public boolean getFree(): this method returns the value of free.

Public void setFree(boolean fr): this method is responsible of setting the free attribute to the ’fr’ passed in parameters.

Public void removePoints(Player p): this method will call the subtractPoints method of the player ’p’ passed in parameter

Public void addPoints(Player p): this method will call the addPoints method of the player ’p’ passed in parameter

Public Box getPusher(Box b): this method will return the first box pushed by the first worker, we keep asking for the previous thing until the previous is equal to null

Public Box getPusher(Worker w): this method will return the first box pushed by the first worker, we keep asking for the previous thing until the previous is equal to null

8.1.3 Direction

- **Responsibility**

Direction is an enumeration that represents the possible ways of a move, it takes the value of either of the following cases: UP, DOWN, RIGHT, LEFT

- **Superclasses**

-

- **Interfaces**

Serializable

- **Attributes**

-

- **Methods**

-

8.1.4 Player

- **Responsibility**

Represents a Player in a Game, the Player class and the Worker class have a bijective relationship, a worker belongs to one player, and a player belongs to one worker.
The player that wins the Game is the one with the highest score.

- **Superclasses**

-

- **Interfaces**

Serializable

- **Attributes**

Points : this attribute is a private integer that stores the current score of the player

Worker: this attribute is private and stores an object of type Worker, its main role is indicating which worker the player belongs to.

- **Methods**

Public void moveWorker(Direction d): this method will call the move method of the worker attribute with the direction 'd' passed in parameter.

Public Worker getWorker(): this method will return the Worker attribute of the Player.

Public setWorker(Worker w): this method sets the Worker attribute to the Worker w passed in the argument of the method.

Public int getPoints(): this method will return the attribute Points of the Player

Public void addPoints(int a): this method will add a to the current value of the Points attribute.

Public void subtractPoints(int a): this method will subtract a from the current value of the Points attribute.

8.1.5 Thing

- **Responsibility**

This is an abstract class that represents any entity that can be placed on the fields.

- **Superclasses**

-

- **Interfaces**

Serializable

- **Attributes**

Field: this attribute is protected of type Fields, it indicates the position of the Thing in the map
Previous: each thing has a Thing type reference which can be set as the pusher for that thing subclass name when a move happens this will be set for all the Boxes and Workers on the push sequence.

- **Methods**

Public Fields getField(): this method will return the current field.

Public void setField(Fields f): this method will set the field attribute to the Fields 'f' passed in the arguments of the method.

Public abstract HitByWorker(Worker w, Direction d): this method will be overridden by all of the descendant of the Thing class

Public abstract HitByBox(Box b, Direction d): this method will be overridden by all of the descendant of the Thing class

8.1.6 Box

- **Responsibility**

Represents a Box on a field. When the game starts, the Map generates Boxes and Designated Squares for them. The main objective of the game is to move these Boxes to the Designated Squares, which will add points to the Players. These boxes can only be moved by Workers.

- **Superclasses**

Thing

- **Interfaces**

Serializable

- **Attributes**
  - **reachedDesignated**: This attribute is used to know when the Box has reached a DesignatedSquare, and when it's not on one. When this goes to true it means the Box is on the DesignatedSquare, and when it's false, then if it was on a DesignatedSquare we know that it moved so we need to remove the points from the Player who pushed it out. The attribute returns a boolean, and is public. By default it's set to false, unless you initialize the Map with a Box on a DesignatedSquare, then the DesignatedSquare changes it to true.

  - **previous**: This attribute is used to know who pushed the Box the last time, it is used in the case when a Worker pushes a Box on an another Worker who pushes a Box on a DesignatedSquare, then the first Worker is the one who gets the points,

therefore we store the previous „Thing" who pushed it. It is a reference to the Thing which may call this object by the **HitByBox** or **HitByWorker** The attribute is public and stores an object of type „Thing" and is null as initial value.

- **field:** This attribute stores the field of the Box, it is used in order to know where this Box is on the Map, as row and column. The attribute is protected and stores an object of type **Fields**, the field is initialised when the Box is created.

- **Methods**
  - **Public Box(Fields f):** This is the constructor of the Box, it takes a Field as an argument and it sets the attribute **field** of the box to this „f" Field, then it sets the attribute **previous** to null, and finally sets the attribute **setFree** of the „Field f" to false.

  - **Public void Die()**: This method is used whenever a Box dies, which is the case when it steps on a **Hole**. When the method is called, it sets the **setFree** attribute of the field to true, and then calls the **Remove**(Thing t) method with (this) argument in order to remove the Box from the Field.

  - **Public boolean HitByBox(Box b, Direction d):** If the method's return value is true it means the move was successful, otherwise it's not. The method is called by the Box (b) in the argument on (this) Box when b is pushed on it. The algorithm work as follows:
    1. Get the neighboring field.
    2. Check the return value of CheckNull on the neighboring field.
    3. If CheckNull is true, and if getSizeOfThings returns 0 then we don't need further check, the Box is moved to the next field, the previous pusher is set to (b) and the method returns true.
    4. If CheckNull is true, and the getSizeOfThings returns 1, it means we have a Thing on the next field, therefor we need to call the same method HitByBox(this, d) on this Thing. If the HitByBox returns false, then we cannot move, therefore the method returns false. If it's true then we can do the move as done previously and the method returns true.
    5. If CheckNull is false, or getSizeOfThings is not 0 or 1, then we call the method HitByBox(this, d) on the Things that are on the field, and if it returns false, the method returns false. If they return true, then we move as done previously and it returns true.

  - **Public boolean HitByWorker(Worker w, Direction d):** If the method's return value is true it means the move was successful, otherwise it's not. The method is called by the Worker (w) in the argument on (this) Box when (w) pushes it. The algorithm works the same as HitByBox(Box b, Direction d) but instead of setting the pusher to the Box (b), we set it to the Worker (w).

8.1.7 Hole

- **Responsibility**

It represents a Hole on a field. If a Worker or Box step on it, they die. A Hole can be turned on/off if it has a Switch, it's done by moving a Box on the Switch to turn it on/off.

- **Superclasses**

Thing

- **Interfaces**

Serializable

- **Attributes**
  - **state**: This attribute is a boolean with public visibility, it's initial value is set to true in the constructor. The attribute is used to specify the state of the Hole, if it is ON (True), or OFF (False). If a Hole is ON then it's active and it kills anyone who steps on it, but if it's OFF then it acts as a normal field, and anyone can step on it.

- **Methods**
  - **Public Hole(Fields f):** This is the constructor of the class, it takes a Field (f) in the argument, sets the field attribute to (f), initializes the state to **true**, and finally sets the **setFree** attribute of the field to **false**.

  - **Public void SetHoleState(boolean state)**: This method is called by a Switch, whenever a Box steps on the Switch, it turns it **ON**, which will turn the state of this Hole to **OFF**. In other words, it sets the **state** attribute to the state given in the argument.

  - **Public boolean HitByBox(Box b, Direction d):** When this method is called for a Hole, it's return value will always be **true**, regardless of the **state** attribute, but if the **state** attribute is **true** then the Box (b) will die, otherwise it can be stepped on as a normal Field.

  - **Public boolean HitByWorker(Worker w, Direction d):** When this methods is called for a Hole, it's return value will always be **true**, regardless of the state attribute, but if the state attribute is **true**, then the Worker (w) will die, otherwise it can be stepped on as a normal Field.

  - **Public boolean getState():** This method returns the value of the attribute **state**.

  - **Public void setState(boolean state):** This method takes a boolean as an argument and sets the attribute **state** to it.

8.1.8 Switch

- **Responsibility**

It represents a Switch on a field. If a Box step on it, the switch's state is changed. Each Switch has a correspoding Hole, it is used to control the state of the Hole. When the Switch is turned ON, the Hole is turned OFF.

- **Superclasses**

Thing

- **Interfaces**

Serializable

- **Attributes**
  - **state**: This attribute is a boolean with public visibility, it's initial value is set to **false** in the constructor. The attribute is used to specify the state of the Switch, if it is ON (True), or OFF (False). If a Switch is OFF then the Hole is active and it kills anyone who steps on it, but if the Switch is ON then the Hole acts as a normal field, and anyone can step on it.

  - **hole***:* This attribute is private, of type Hole. It stores a Hole object, which corresponds to (this) Switch.

- **Methods**
  - **Public Switch(Hole h, Fields f):** This is the constructor of the class, it takes a Field (f) and a Hole (h) in the argument, sets the **field** attribute to (f), initializes the **state** to **false**, and finally sets the **hole** attribute to the Hole given in the argument.

  - **Public Hole getHole()**: This method returns the Hole object stored in the **hole** attribute.

  - **Public boolean HitByBox(Box b, Direction d):** When this method is called for a Switch, it's return value will always be **true**, regardless of the **state** attribute. When the Switch is hit by a Box, first it will set the state of the switch to **true** (ON), and then set the state of the Hole corresponding to it to **false** (OFF), and return **true**.

  - **Public boolean HitByWorker(Worker w, Direction d):** When this methods is called for a Hole, it's return value will always be **true**, regardless of the state attribute. When the Switch is hit by a Worker, nothing happens.

  - **Public boolean getState():** This method returns the value of the attribute **state**.

  - **Public void setState(boolean state):** This method takes a boolean as an argument and sets the attribute **state** of the switch to it, and then sets the attribute **state** of the hole to the opposing value. (Switch ON -> Hole OFF)

8.1.9 Wall

- **Responsibility**

It represents a Wall on a field, whose only responsibility is to block the move of a Worker or a Box.

- **Superclasses**

Thing

- **Interfaces**

Serializable

- **Attributes**

- **Methods**
  - **Public Wall(Fields f):** This is the constructor of the class, it takes a Field (f) in the argument, sets the **field** attribute to (f), and sets the **setFree** attribute of the field to **false.**

  - **Public boolean HitByBox(Box b, Direction d):** When this method is called, it will always return false.

  - **Public boolean HitByWorker(Worker w, Direction d):** When this method is called, it will always return false.

8.1.10 Game

- **Responsibility**

  It handles the overall process of the game start and end. Also it manages the actual Players that are controlling the Workers.

- **Superclasses**

  -

- **Interfaces**

  Serializable

- **Attributes**

  -map: This attribute's type is a Map, it stores the map created in Start game function.
  -players: this attribute is an array list of type Player, it stores the players objects.

- **Methods**
  - **Public void addPlayer(p:Player):** adds the Player p to Players array.
  - **Public void startGame ():** this function asks the user at the beginning for the number of players and stores them in the Players ArrayList attribute by using the Addplayer function. after that, it starts filling the map by choosing an exact index (Row, column) with a specific object from a list of things (1-Worker 2-Box 3-Hole with switch 4-Hole without switch 5-wall 6-DesignatedSquare).

- **Public void endGame():** ends the game when it remains only one worker on the map.

8.1.11 Map

- **Responsibility**

Map is a collection of Fields and is responsible to Stores the Fields of the game which are the blocks making the grid of the Map.

- **Superclasses**
  -

- **Interfaces**
  - Serializable

- **Attributes**

  -fields: this attribute is 2-dimensional array of type Fields, each Field stores an array of thing of size 2.
  -size: this attribute's type is integer, it stores the size of the map (the map is a square matrix so it means that size = Number of row=Number of column) .

- **Methods**
  - **Public void map (int size) :** this is the constructor of the map class, it takes size as a parameters, assigns it to size attribute and starts creating the fields. When it creates the fields, it assigns a specific column and row value for each field .
  - **Public void setMapFields (int s) :** it takes size attribute as a parameters and starts creating the fields. While creating the fields, it assigns a specific column and row value for each field.
  - **Public void addThing(int i,int j,Thing t) :**it takes a specific index(i,j), instance of thing as parameters and store the thing in the array of things which is stored in the fields[i][j]. Adds Things **t** to the things array of the Fields on index(i,j) of the 2d array.
  - **Public void setFields(int I,int j,Thing t):** it takes a specific index(i,j), instance of thing as parameters and store the thing in the index 0 of the array of thing which is stored inside the fields[i][j].
  - **Public void setSize(int size)**: assign the parameter size to the size attribute of the map class.
  - **Public int getSize():**it returns the size of the map (so it means that size = Number of row=Number of column**)**

8.1.12 Fields

- **Responsibility**

2. Fields represent the squares in the task description, the other interactive elements of the game will be associated with these and in some way placed on them. Their responsibility is to be able to get information about neighboring fields and either accepting or removing objects from themselves. Everything element must have one Field

and uses it in almost every case of the game to fulfill its requirements. It contains at most two Thing objects which can be accessed through the Getters and Setters.

- **Superclasses**
  -

- **Interfaces**
  - Serializable

- **Attributes**
  -map :Map variable which will be shared(it will be the same map for) by the 2d array of fields
  - col : this attribute's type is integer, it stores the number of column.
  -row: this attribute's type is integer, it stores the number of rows.
  -free : this attribute's type is boolean, it indicates if I can put a box or a worker on the fields because we should handle the case when the user puts a worker and a box on the same field
  -things[]:this attribute's type is Thing, it stores the things on the field.
  -sizeOfThing: this attribute's type is integer, it shows how many things are there on the field.

- **Methods**
  - **Public void field (Map map) :** this is the constructor of the fields class, it assigns the map of the fields instance,set the sizeOfThing to 0,free to true and creates Things array.
  - **Public void setMapFields (int s) :** it takes size attribute as a parameters and starts creating the fields. When it creates the fields, it assigns a specific column and row value for each field.
  - **Public void set map(Map map):** it assign the map attribute with this map parameter
  - **Public getFree(Boolean free)** : it returns the state of the free attribute.
  - **Public void setFree(Boolean free)**: assign the free attribute as this free parameter.
  - **Public int getSizeOfThings():**it returns the sizeOfThing attribute.
  - **public void addThing(Thing t):** Adds a thing to its things[] after deciding which position is empty
  - **public void Accept(Thing t) :** Accepts a thing to its things array
  - **public boolean CheckNull() :** Returns true if the
  - **public void SetNeighbor(Direction d, Fields f)** Sets neighboring Field to be **f** in a given direction **d**
  - **public Fields GetNeighbor(Direction d) :** Gets (returns) neighbor in direction **d**
  - **public Thing[] GetThing() :** Returns a reference to the array of **things** that are on the field(this.things)
  - **public void SetThing(int i, Thing t) :** sets a things array index **I** to be **t** passed as parameter
  - **public int getCol() :** gets the column rank (value) of the field
  - **public void setCol(int col) :** sets the column rank (value) of the field to be **col**
  - **public int getRow() :** gets the row rank (value) of the field
  - **public void setRow(int row):** sets the row rank (value) of the field to be **row**

- **public String toString() :** prints out the **row** and **column** values in a string with a given format

8.2 Detailed plan of testing

8.2.1 Worker Worker

- **Description**

This test case shows the interaction between two workers, the first Worker moves to the Field containing the second Worker. In this scenario the first Worker is placed on the field [1][1], and the second on the field [1][2] (The borders have Walls, therefore we start with index 1).

- **Unit of functionality to be tested, possible failures**

In this scenario, the interaction between two Workers is tested. The expected output is that nothing happens when a Worker is hit by another Worker. This test can fail if the Workers moved to next field.

- **Input**

load scenario 1
game state
move worker right
game state

- **Output**

Scenario 1 loaded

Game size 10
class Worker [1][1]
class Worker [1][2]
Player 0 points: 0
Player 1 points: 0

Move executed (right)

Game size 10
class Worker [1][1]
class Worker [1][2]
Player 0 points: 0
Player 1 points: 0

8.2.2 Worker Box

- **Description**

This test cases shows the interaction between a Worker and a Box. The Worker moves to the field containing the Box. The Worker is placed on the field [1][1] and the Box is on the field [1][2].

- **Unit of functionality to be tested, possible failures**

In this scenario, the basic move interaction between a Worker and a Box is tested. The expected output would be for both of them to move one field according to the direction given. This test can fail if the Box or the Worker didn't move.

- **Input**

load scenario 2
game state
move worker right
game state

- **Output**

Scenario 2 loaded

Game size 10
class Worker [1][1]
class Box [1][2]
Player 0 points: 0

Move executed (right)

Game size 10
class Worker [1][2]
class Box [1][3]
Player 0 points: 0

8.2.3 Worker EmptySquare

- **Description**

This test cases shows the interaction between a Worker and an empty field. The Worker moves to the field. The Worker is placed on the field [1][1] and is surrounded by empty fields on [1][2] , [2][1].

- **Unit of functionality to be tested, possible failures**

In this scenario, the basic move interaction between a Worker and an empty field is tested. The expected output would be for the Worker to move one field according to the direction given. This test can fail if the Worker didn't move.

- **Input**

load scenario 3
game state
move worker right
game state

- **Output**

Scenario 3 loaded

Game size 10
class Worker [1][1]
Player 0 points: 0

Move executed (right)

Game size 10
class Worker [1][2]
Player 0 points: 0

8.2.4 Worker Switch

- **Description**

This test cases shows the interaction between a Worker and a Switch. The Worker moves to the field containing the Switch. The Worker is placed on the field [1][1] and the Switch is on the field [1][2], the state of the switch is false by default.

- **Unit of functionality to be tested, possible failures**

In this scenario, the basic move interaction between a Worker and a Switch is tested. The expected output would be for the Worker to move to the field contaning the Switch, but the Switch state doesn't change. This test can fail if the Worker didn't move, or it moved and the state of Switch changed.

- **Input**

load scenario 4
game state
move worker right
game state

- **Output**

Scenario 4 loaded

Game size 10
class Worker [1][1]
class Switch [1][2]
Switch state: false
Player 0 points: 0

Move executed (right)

Game size 10
class Switch [1][2]
class Worker [1][2]
Switch state: false
Player 0 points: 0

8.2.5 Worker DesignatedSquare

- **Description**

This test cases shows the interaction between a Worker and a DS. The Worker moves to the field containing the DS. The Worker is placed on the field [1][1] and the DS is on the field [1][2], the Worker has 0 points by default.

- **Unit of functionality to be tested, possible failures**

In this scenario, the basic move interaction between a Worker and a DS is tested. The expected output would be for the Worker to move to the field contaning the DS, but he shouldn't get any points. This test can fail if the Worker didn't move, or it moved and received points for stepping on a DS.

- **Input**

load scenario 5
game state

move worker right
game state

- **Output**

Scenario 5 loaded

Game size 10
class Worker [1][1]
class DesignatedSquare [1][2]
Player 0 points: 0

Move executed (right)

Game size 10
class DesignatedSquare [1][2]
class Worker [1][2]
Player 0 points: 0

8.2.6 Worker Hole

- **Description**

This test cases shows the interaction between a Worker and a Hole. The Worker moves to the field containing the Hole. The Worker is placed on the field [1][1] and the Hole is on the field [1][2], by default the Hole is turned ON (state=true).

- **Unit of functionality to be tested, possible failures**

In this scenario, the basic move interaction between a Worker and a Hole is tested. The expected output would be for the Worker to move to the field contaning the Hole, and then die. This test can fail if the Worker didn't move, or it moved and is still alive.

- **Input**

load scenario 6
game state
move worker right
game state

- **Output**

Scenario 6 loaded

Game size 10
class Worker [1][1]
class Hole [1][2]
Player 0 points: 0

Move executed (right)

Game size 10
class Hole [1][2]
Player 0 points: 0

8.2.7 Worker Wall

- **Description**

This test cases shows the interaction between a Worker and a Wall. The Worker moves to the field containing the Wall. The Worker is placed on the field [1][1] and is surrounded by Walls on [0][1], [1][0], [1][2]. (Borders have walls)

- **Unit of functionality to be tested, possible failures**

In this scenario, the basic move interaction between a Worker and a Wall is tested. The expected output would be for the Worker to not be able move to the field contaning the Wall, since it blocks the move. This test can fail if the Worker was able to move to the field containing the Wall.

- **Input**

load scenario 7
game state
move worker right
game state

- **Output**

Scenario 7 loaded
Game size 10
class Worker [1][1]
class Wall [1][2]
Player 0 points: 0
Move executed (right)
Game size 10
class Worker [1][1]
class Wall [1][2]
Player 0 points: 0

8.2.8 Box worker

- **Description:**

  This test case shows the interaction between Box and two Workers, the first worker will move to the field containing the box and inevitably make the box push the second worker.

- **Unit of functionality to be tested, possible failures**

  In this scenario, the basic move interaction between a box and worker is tested. The expected output is for the second worker to move accordingly to the box. The purpose of first worker is to push the box. This test can fail if the second worker or the box did not move.

- **Input :**

  load scenario 8
  game state
  move worker right
  game state

- **Output :**

  Scenario 8 loaded

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Worker [1][3]
  Player 0 points: 0

Player 1 points: 0

Move executed (right)

Game size 10
class Worker [1][2]
class Box [1][3]
class Worker [1][4]
Player 0 points: 0
Player 1 points: 0

8.2.9 Box Box

- **Description:** this test case shows the interaction between two boxes and a worker, worker will move to the field containing the first box and inevitably make the first box push the second box.
- **Unit of functionality to be tested, possible failures**
  In this scenario, the basic move interaction between two boxes is tested. The expected output is for the second box to move accordingly (In the direction of) to the second box. The purpose of first worker is to push the box and start the movement because a box cannot move itself. This test can fail if the second box or the first box did not move.
- **Input:**
  load scenario 9
  game state
  move worker right
  game state
- **Output:**

  Scenario 9 loaded

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Box [1][3]
  Player 0 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][2]
  class Box [1][3]
  class Box [1][4]
  Player 0 points: 0

8.2.10 Box Empty Square

- **Description**

This test case shows the interaction between a box and a empty square, worker will move to the field containing the box and inevitably make the box move in the same direction .

- **Unit of functionality to be tested, possible failures**
  In this scenario, the basic move of a box is tested. The expected output would be for the box and the worker to move to the appropriate field. The purpose of worker is to push the box and start the movement because a box cannot move itself. This test can fail if the box or the worker did not move to the appropriate field.
  Input:
  load scenario 10
  game state
  move worker right
  game state
  move worker right
  game state
- **Output:**
  Scenario 10 loaded

  Worker created
  Worker: [1][1]

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  Player 0 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][2]
  class Box [1][3]
  Player 0 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][3]
  class Box [1][4]
  Player 0 points: 0

8.2.11 Box Switch

- **Description:** this test case shows the interaction between a box and a Switch, worker will move to the field containing the box and inevitably make the box move onto the switch and turn it on (the default state of the switch is off)
- **Unit of functionality to be tested, possible failures**
  In this scenario, the basic move interaction between box and a switch is tested. The expected output would be for the box to move to the field containing the switch and for the state of the switch to change. The purpose of first worker is to push the box and start the movement because a box cannot move itself. This test can fail if the box did

not move onto the same field as the switch or the inner state of the switch did not change .

- **Input:**
  load scenario 11
  game state
  move worker right
  game state
- **Output:**
  Scenario 11 loaded

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Switch [1][3]
  class Hole [2][2]
  Player 0 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][2]
  class Switch [1][3]
  class Box [1][3]
  class Hole [2][2]
  Player 0 points: 0

8.2.12 Box Designated Square

- **Description:** this test case shows the interaction between a box and a designated square, worker will move to the field containing the box and inevitably make the box move onto the designated square and add points to the worker.
- **Unit of functionality to be tested, possible failures**
  In this scenario, the basic move interaction between box and a designated square is tested. The expected output would be for the box to move to the field containing the designated square and the DS will add points to the worker. The purpose of first worker is to push the box and start the movement because a box cannot move itself. This test can fail if the box did not move onto the same field as the DS or the points of the worker did not change .
- **Input:**
  load scenario 12
  game state
  move worker right
  game state
- **Output:**
  Scenario 12 loaded

Game size 10
class Worker [1][1]
class Box [1][2]
class Designated Square[1][3]
Player 0 points: 0

Move executed (right)

Game size 10
class Worker [1][2]
class Box [1][3]
class Designated Square[1][3]
Player 0 points: 5

8.2.13 Box Hole

- **Description**
  This test case shows the interaction between a box and hole, worker will move to the field containing the box and inevitably push the box onto the hole which will discard the box.
- **Unit of functionality to be tested, possible failures**
  In this scenario, the interaction of a box and a hole is tested. The expected output would be for the worker to move to the appropriate field and for the box to die(call of the die function) . The purpose of worker is to push the box and start the movement because a box cannot move itself. This test can fail if the box or the worker did not move or the box did not get deleted after the move.
- **Input:**
  load scenario 13
  game state
  move worker right
  game state
  Output:
  Scenario 13 loaded

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Hole [1][3]
  Player 0 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][2]
  class Hole [1][3]

Player 0 points: 0

8.2.14 Box Wall

- **Description**
  This test case shows the interaction between a box and wall, worker will try move to the field containing the box and make the box move in the same direction, but the Wall will not permit the box to move ,thus the whole move should result in no change.
- **Unit of functionality to be tested, possible failures**
  In this scenario, the basic move of a box is tested as well as the immovability of the wall. The expected output would be for the box and the worker to try to ,but result in no change. The purpose of worker is to push the box and start the movement because a box cannot move itself. This test can fail if the box or the worker moved.
- **Input:**
  load scenario 14
  game state
  move worker right
  game state

  Output:
  Scenario 14 loaded

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Wall [1][3]
  Player 0 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Wall [1][3]
  Player 0 points: 0

8.2.15 Worker Box (Switch+Box)

- **Description**
  This test case shows the interaction between a box and another box and a switch on the same field, worker will try move to the field containing the box and make the box move in the same direction and push the other box out of the switch and replace the box with first box.
- **Unit of functionality to be tested, possible failures**
  In this scenario, the interaction between two boxes and a switch will be tested as well as score management of player. The expected output would be for the boxes and the

worker to move and 5 points added to player 0. This test can fail if the box or the worker did not move or the points of the player didn't change.

Input:
load scenario 15
game state
move worker right
game state

Output:
Scenario 15 loaded

Game size 10
class Worker [1][1]
class Box [1][2]
class Switch [1][3]
class Box [1][3]
Player 0 points: 0

Move executed (right)

Game size 10
class Worker [1][2]
class Switch [1][3]
class Box [1][3]
class Box [1][4]
Player 0 points: 5

8.2.16 Worker Box Worker Box Wall

- **Description**
  This test case shows what happens when a worker is squeezed between 2 boxes where one is pushed by another worker, first worker will push the first box and kill the second worker.
- **Unit of functionality to be tested, possible failures**
  In this scenario, the interactions between a sequence of boxes and workers will be tested. The expected output would be for the first box and the first worker to move and kill the second worker . This test can fail if the first box or the first worker did not move or the second worker did not die.

Input:
load scenario 16
game state
move worker right
game state

Output:
Scenario 16 loaded

Game size 10
class Worker [1][1]
class Box [1][2]
class Worker [1][3]

class Box [1][4]
class Wall [1][5]
Player 0 points: 0
Player 1 points: 0

Move executed (right)

Game size 10
class Worker [1][2]
class Box [1][3]
class Box [1][4]
class Wall [1][5]
Player 0 points: 0
Player 1 points: 0

8.2.17 Worker Box Box Worker Box DesignatedSquare

- **Description**
  This test case shows what happens when a worker and box sequence is pushed as a line where one is pushed by another, first worker will push the first box and the rest will move accordingly and the points will be added to the first(defined in description) player(worker).
- **Unit of functionality to be tested, possible failures**
  In this scenario, the interactions between a sequence of boxes and workers will be tested. The expected output would be for all the boxes and the workers to move in the same direction and add points to the first worker(according player) . This test can fail if the boxes or the workers did not move or the second worker was awarded instead of the first.

- **Input:**
  load scenario 17
  game state
  move worker right
  game state

  Output:
  Scenario 17 loaded

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Box [1][3]
  class Worker [1][4]
  class Box [1][5]
  class Designated Square [1][6]
  Player 0 points: 0
  Player 1 points: 0

Move executed (right)

Game size 10
class Worker [1][2]
class Box [1][3]
class Box [1][4]
class Worker [1][5]
class Box [1][6]
class Designated Square [1][6]
Player 0 points: 5
Player 1 points: 0

8.2.18 Worker Box Box Worker EmptySquare

- **Description**
  This test case shows what happens when a worker and box sequence is pushed as a line where one is pushed by another, first worker will push the first box and the rest will move accordingly.
- **Unit of functionality to be tested, possible failures**
  In this scenario, the interactions between a sequence of boxes and workers will be tested. The expected output would be for all the boxes and the workers to move in the same direction . This test can fail if the boxes or the workers did not move or some of them moved in a different way.

- **Input:**
  load scenario 18
  game state
  move worker right
  game state

- **Output:**
  Scenario 18 loaded

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Box [1][3]
  class Worker [1][4]
  Player 0 points: 0
  Player 1 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][2]
  class Box [1][3]
  class Box [1][4]

class Worker [1][5]
Player 0 points: 0
Player 1 points: 0

8.2.19 Worker Box Worker (Box+DesignatedSquare) EmptySquare

- **Description**
  This test case shows what happens when a worker and box sequence is pushed as a line where one is pushed by another, first worker will push the first box and the rest will move accordingly and when the second Worker inevitably moves the box out of the DS he will lose points .
- **Unit of functionality to be tested, possible failures**
  In this scenario, the interactions between a sequence of boxes and workers will be tested. The expected output would be for all the boxes and the workers to move in the same direction and the second worker (player) to lose points. This test can fail if the boxes or the workers did not move or some of them moved in a different way or the points of the players don't change.

- **Input:**
  load scenario 19
  game state
  move worker right
  game state
- **Output:**
  Scenario 19 loaded

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Worker [1][3]
  class Box [1][4]
  class Designated Square [1][4]
  Player 0 points: 0
  Player 1 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][2]
  class Box [1][3]
  class Worker [1][4]
  class Box [1][5]
  class Designated Square [1][4]
  Player 0 points: 0
  Player 1 points: -5

8.2.20 Worker Box Worker(Box +DesignatedSqure) Box DesignatedSquare

- **Description**

This test case shows what happens when a worker and box sequence is pushed as a line where one is pushed by another, first worker will push the first box and the rest will move accordingly and when the second Worker inevitably moves the box out of the DS second worker will lose points and at the end of the line when the third box goes onto the second designated square therefore the first worker will gain points.

- **Unit of functionality to be tested, possible failures**

  In this scenario, the interactions between a sequence of boxes and workers will be tested. The expected output would be for all the boxes and the workers to move in the same direction and the second worker (player) to lose points while the first worker to gain points. This test can fail if the boxes or the workers did not move or some of them moved in a different way or the points of the players don't change.

- **Input:**

  load scenario 20
  game state
  move worker right
  game state

  Output:
  Scenario 20 loaded

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Worker [1][3]
  class Box [1][4]
  class Designated Square [1][4]
  class Box [1][5]
  class Designated Square [1][6]
  Player 0 points: 0
  Player 1 points: 0

  Move executed (right)

  class Worker [1][2]
  class Box [1][3]
  class Worker [1][4]
  class Designated Square [1][4]
  class Box [1][5]
  class Box [1][6]
  class Designated Square [1][6]
  Player 0 points: 5
  Player 1 points: -5

8.2.21 Worker Box (Worker+DesignatedSquare) Wall

- **Description**

  This test case shows what happens when a worker and box sequence is pushed as a line where one is trying to push another and when the second Worker is pushed onto

or squeezed between wall and the box also at the same time the box will move onto the DS therefore the first worker will gain points.

- **Unit of functionality to be tested, possible failures**
  In this scenario, the interactions between a sequence of boxes and workers will be tested. The expected output would be for all the boxes and the first worker to move in the same direction and the second worker (player) to die while the first worker will gain points. This test can fail if the boxes or the workers did not move or some of them moved in a different way or the points of the first player didn't change.

- **Input:**
  load scenario 21
  game state
  move worker right
  game state
  Output:
  Scenario 21 loaded

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Worker [1][3]
  class Designated Square [1][3]
  class Wall [1][4]
  Player 0 points: 0
  Player 1 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][2]
  class Box [1][3]
  class Designated Square [1][3]
  class Wall [1][4]
  Player 0 points: 5

8.2.22 Worker Box (Switch + Worker) Hole

- **Description**
  This test case shows interaction between Switch and a hole when there are 2 box and worker type Things on the Fields .The initial **state** variables of the Switch and the linked hole is off.
- **Unit of functionality to be tested, possible failures**
  In this scenario, the interactions between a sequence of boxes and workers will be tested while considering the internal switch of the hole will be at first off then on because the box will step on the Switch a few milliseconds before the second worker will step on the Hole. The expected output would be for the box and the first worker to move and the second worker (player) to die when stepping on the hole. This test can fail if the box or the workers did not move or some of them moved in a different way

or state of the Hole didn't change which will then prevent the second worker from dying but in this case it is expected to die .

- **Input:**
  load scenario 22
  game state
  move worker right
  game state
- **Output:**
  Scenario 22 loaded

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Switch [1][3]
  class Worker [1][3]
  class Hole [1][4]
  Player 0 points: 0
  Player 1 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][2]
  class Box [1][3]
  class Switch [1][3]
  class Hole [1][4]
  Player 0 points: 0

8.2.23 Worker Box (DesignatedSquare+Worker) Hole

- **Description**
  This test case shows a complex interaction between multiple objects. Worker moves the Box that moves a Worker that is placed on a DesignatedSquare to a Hole
  The first Worker is placed on the field [1][1], the first Box on field [1][2], the second Worker on field[1][3], DesignatedSquare on field [1][3], and Hole on field[1][4]

- **Unit of functionality to be tested, possible failures**
  In this scenario, a complex functionality is tested, sub-tests are tested(Worker-Box, Box-Worker, Box-DesignatedSquare, Worker-Hole)
  The expected output is each that all the objects, except of the Hole and DesignatedSquare, would move to the right, and the second worker should die. (all the sub-tests return True)
  This test can fail if the second Worker does not die (because the state of the Hole is on) or/and either one of the worker or boxes don't move to the right, and/or the second player's points change and/or the first player does not get added points.

- **Input**
  load scenario 23

game state
move worker right
game state

- **Output**
  Scenario 23 loaded
  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class DesignatedSquare [1][3]
  class Worker [1][3]
  class Hole [1][4]

  Player 0 points: 0
  Player 1 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][2]
  class Box [1][3]
  class DesignatedSquare [1][3]
  class Hole [1][4]

  Player 0 points: 5
  Player 1 points: 0

8.2.24 Worker Box (Switch+Box) Hole

- **Description**
  This test case shows a complex interaction between multiple objects. Worker moves the Box that moves a second Box that is placed on a Switch(which belongs to the hole on field[1][4]) to a Hole
  The first Worker is placed on the field [1][1], the first Box on field [1][2], the second Box on field[1][3], Switch on field [1][3], and Hole on field[1][4]

- **Unit of functionality to be tested, possible failures**
  In this scenario, a complex functionality is tested, sub-tests are tested(Worker-Box, Box-Switch, Box-Hole)
  The expected output is each that all the objects, except of the Hole and Switch, would move to the right. (all the sub-tests return True)
  This test can fail if the second Box disappears (because the switch is on so its related hole is off) or/and either one of the worker or boxes don't move to the right, and/or the player's points change.

- **Input**
  load scenario 24
  game state

move worker right
game state

- **Output**
  Scenario 24 loaded
  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Box [1][3]
  class Switch [1][3]
  class Hole [1][4]

  Player 0 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][2]
  class Box [1][3]
  class Box [1][4]
  class Switch [1][3]
  class Hole [1][4]

  Player 0 points: 0

8.2.25 Worker Box (Switch+Box) Worker Hole

- **Description**

  This test case shows a complex interaction between multiple objects. Worker moves the Box that moves a second Box that is placed on a Switch(which belongs to the hole on field[1][5], the second Box pushes the second Worker to the Hole.
  The first Worker is placed on the field [1][1], the first Box on field [1][2], the second Box on field[1][3], Switch on field [1][3], Worker on field [1][4] Hole on field[1][5]

- **Unit of functionality to be tested, possible failures**
  In this scenario, a complex functionality is tested, sub-tests are tested(Worker-Box, Box-Switch, Box-Worker, Worker-Hole)
  The expected output is each that all the objects, except of the Hole and Switch, would move to the right.
  This test can fail if the second Worker dies (because the switch is on so its related hole is off) or/and either one of the workers or boxes don't move to the right, and/or the players' points change.
- **Input**
  load scenario 25
  game state
  move worker right
  game state
- **Output**
  Scenario 25 loaded

Game size 10
class Worker [1][1]
class Box [1][2]
class Box [1][3]
class Switch [1][3]
class Worker[1][4]
class Hole [1][5]

Player 0 points: 0
Player 1 points: 0

Move executed (right)

Game size 10
class Worker [1][2]
class Box [1][3]
class Box [1][4]
class Switch [1][3]
class Worker[1][5]
class Hole [1][5]

Player 0 points: 0
Player 1 points: 0

8.2.26 Worker Box Box Worker Box (DesignatedSquare, Worker) Hole

- **Description**
  This test case shows a complex interaction between multiple objects. Worker moves the Box that moves a Box placed on a designatedSquare, the second Box pushes a second Worker that pushes a third Worker.
  The first Worker is placed on the field [1][1], the first Box on field [1][2], both the second Box on field [1][3], the second Worker on field [1][4], the third Box on field [1][5], both the designated square and third Worker on field [1][6], and the Hole on field [1][7].

- **Unit of functionality to be tested, possible failures**
  In this scenario, a complex functionality is tested, sub-tests are tested, (Worker-Box, Box-Box, Box-DesignatedSquare, Box-Worker, Worker-Hole)
  The expected output is each object, except of the Hole and DesignatedSquare, should move.
  This test can fail the third worker does not die, and/or an object except of both the hole and designated square remains at their place, and/or the second players points change

- **Input**
  load scenario 26
  game state
  move worker right

game state

- **Output**
  Scenario 27 loaded
  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Box [1][3]
  class Worker [1][4]
  class Box [1][5]
  class DesignatedSquare[1][6]
  class Worker [1][6]
  class Hole [1][7]

  Player 0 points: 0
  Player 1 points: 0
  Player 2 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][2]
  class Box [1][3]
  class Box [1][4]
  class Worker [1][5]
  class Box [1][6]
  class DesignatedSquare[1][6]
  class Hole [1][7]

  Player 0 points: 5
  Player 1 points: 0
  Player 2 points:0

8.2.27 Worker Box Box Worker (Box DesignatedSquare) Worker Hole

- **Description**
  This test case shows a complex interaction between multiple objects. Worker moves the Box that moves a second Box that pushes a second Worker, the second Worker pushes the third Box that is placed on a designatedSquare, the third Box pushes the third Worker to Hole.
  The first Worker is placed on the field [1][1], the first Box on field [1][2], the second Box on field[1][3], Worker on field [1][4] both the third Box and the designatedSquare on field [1][5], the third Worker on field [1][6], and the Hole on field [1][7]

  Unit of functionality to be tested, possible failures

In this scenario, a complex functionality is tested, sub-tests are tested, (Worker-Box, Box-Box, Box-DesignatedSquare, Box-Worker, Worker-Hole)
The expected output is each object remains at its original field because the sub-test Box-Wall will return false thus all of the other sub-test will return false, therefore the move will not occur.
This test can fail the third worker does not die, and/or an object except of both the hole and designated square remains at their place, and/or the second players points change

- **Input**
  load scenario 27
  game state
  move worker right
  game state

- **Output**
  Scenario 27 loaded
  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Box [1][3]
  class Worker [1][4]
  class Box [1][5]
  class DesignatedSquare[1][5]
  class Worker [1][6]
  class Hole [1][7]

  Player 0 points: 0
  Player 1 points: 0
  Player 2 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][2]
  class Box [1][3]
  class Box [1][4]
  class Worker [1][5]
  class Box [1][6]
  class DesignatedSquare[1][5]
  class Hole [1][7]

  Player 0 points: -5
  Player 1 points: 0
  Player 2 points:0

8.2.28 Worker Box (Box + DesignatedSquare) Worker Worker

- **Description**

This test case shows a complex interaction between multiple objects. Worker moves the Box that moves a Box placed on a designatedSquare, the second Box pushes a second Worker that pushes a third Worker.
The first Worker is placed on the field [1][1], the first Box on field [1][2], both the second Box and the designatedSquare on field [1][3], the second Worker on field [1][4], the third Worker is placed on field [1][5].

- **Unit of functionality to be tested, possible failures**
  In this scenario, a complex functionality is tested, sub-tests are tested, (Worker-Box, Box-Box, Box-DesignatedSquare, Box-Worker, Worker-Worker)
  The expected output is each object remains at its original field because the sub-test Box-Wall will return false thus all of the other sub-test will return false, therefore the move will not occur.
  This test can fail if any objects changes its field or any player gets extra or minus points

- **Input**
  load scenario 28
  game state
  move worker right
  game state

- **Output**
  Scenario 28 loaded
  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Box [1][3]
  class DesignatedSquare [1][3]
  class Worker [1][4]
  class Worker [1][5]

  Player 0 points: 0
  Player 1 points: 0
  Player 2 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Box [1][3]
  class DesignatedSquare [1][3]
  class Worker [1][4]
  class Worker [1][5]

  Player 0 points: 0
  Player 1 points: 0
  Player 2 points:0

8.2.29 Worker Box (Box + DesignatedSquare) Worker Box Wall

- **Description**
  This test case shows a complex interaction between multiple objects. Worker moves the Box that moves a Box placed on a designatedSquare, the second Box pushes a second Worker that pushes a third Box to a Wall.
  The first Worker is placed on the field [1][1], the first Box on field [1][2], both the second Box and the designatedSquare on field [1][3], the second Worker on field [1][4], the third Box is placed on field [1][5], and the Wall on field [1][6].

- **Unit of functionality to be tested, possible failures**
  In this scenario, a complex functionality is tested, sub-tests are tested, (Worker-Box, Box-Box, Box-DesignatedSquare, Box-Worker, Box-Wall)
  The expected output is each object remains at its original field because the sub-test Box-Wall will return false thus all of the other sub-test will return false, therefore the move will not occur.
  This test can fail if any objects changes its field or any player gets extra or minus points
- **Input**
  load scenario 29
  game state
  move worker right
  game state
- **Output**
  Scenario 29 loaded

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Box [1][3]
  class DesignatedSquare [1][3]
  class Worker [1][4]
  class Box [1][5]
  class Wall [1][6]

  Player 0 points: 0
  Player 1 points: 0

  Move executed (right)

  Game size 10
  class Worker [1][1]
  class Box [1][2]
  class Box [1][3]
  class DesignatedSquare [1][3]
  class Worker [1][4]
  class Box [1][5]

class Wall [1][6]

Player 0 points: 0
Player 1 points: 0

8.3 Plans of the supporting programs

There is no need for a supporting program just any java compatible IDE will be able to run the code and tests.

8.4 Protocol

Start (date & time)	Duration (hours)	Performer(s) name	Activity description
[06.04.2018]	3 hours	Khouloud Khezami	Class Description: Worker, DesignatedSquare, Direction, Player, Thing
[08.04.2018]	3 hours	Khezami Khouloud	Test cases from 23 till 29
[06.04.2018] 18:00	3 hours	Rovshan Shirinli, M'ahdi Karray	Test cases from 8 till 13 Class description Map, Fields ,Game
[08.04.2018] 21:00	3 hours	Rovshan Shirinli, M'ahdi Karray	Test cases from 13 till 25
[06.04.2018] 17:00	5 hours	Houssam Mehdi	Class description for the classes Box, Wall, Switch, Hole and Test cases from 1 to 7

10. Prototype program

10.1 Deployment guide

10.1.1 List of files

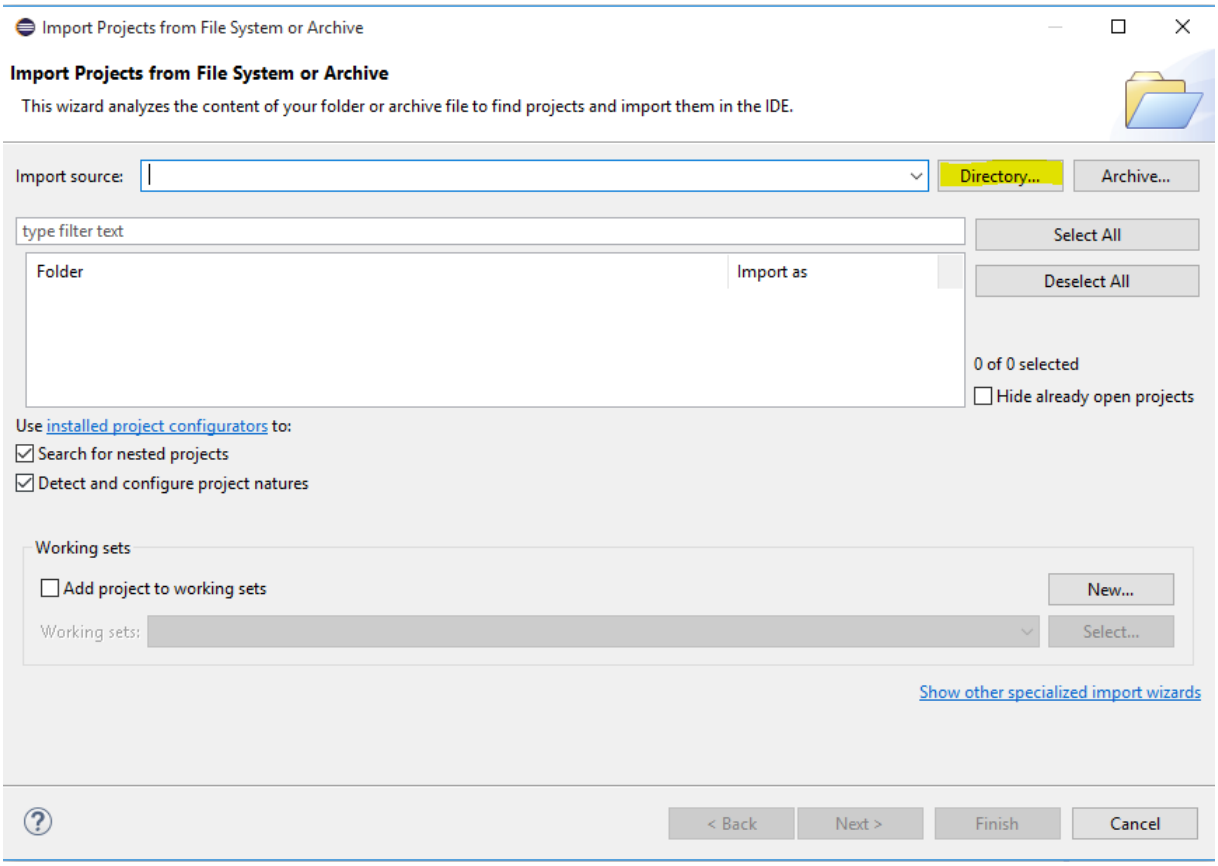
File name	Size	Date	Content
Box.java	4kb	4/23/2018 2:37AM	Box class
Worker.java	4kb	04/23/2018 2:37AM	Worker class
DesignatedSquare.java	3kb	4/23/2018 2:37AM	DS class
Direction.java	1kb	4/23/2018 2:37AM	Enum Directions
Fields.java	3kb	4/23/2018 2:37AM	Fields class
Game.java	9kb	4/23/2018 2:37AM	Game class
Hole.java	1kb	4/23/2018 2:37AM	Hole class
Main.java	18kb	4/23/2018 2:37AM	Main with all the tests
Map.java	2kb	4/19/2018 2:37AM	Map class
Player.java	2kb	4/19/2018 2:37AM	Player class
Switch.java	1kb	4/19/2018 2:37AM	Switch class
Thing.java	1kb	4/19/2018 2:37AM	Thing class
Wall.java	1kb	4/19/2018 2:37AM	Wall class

10.1.2 Compilation

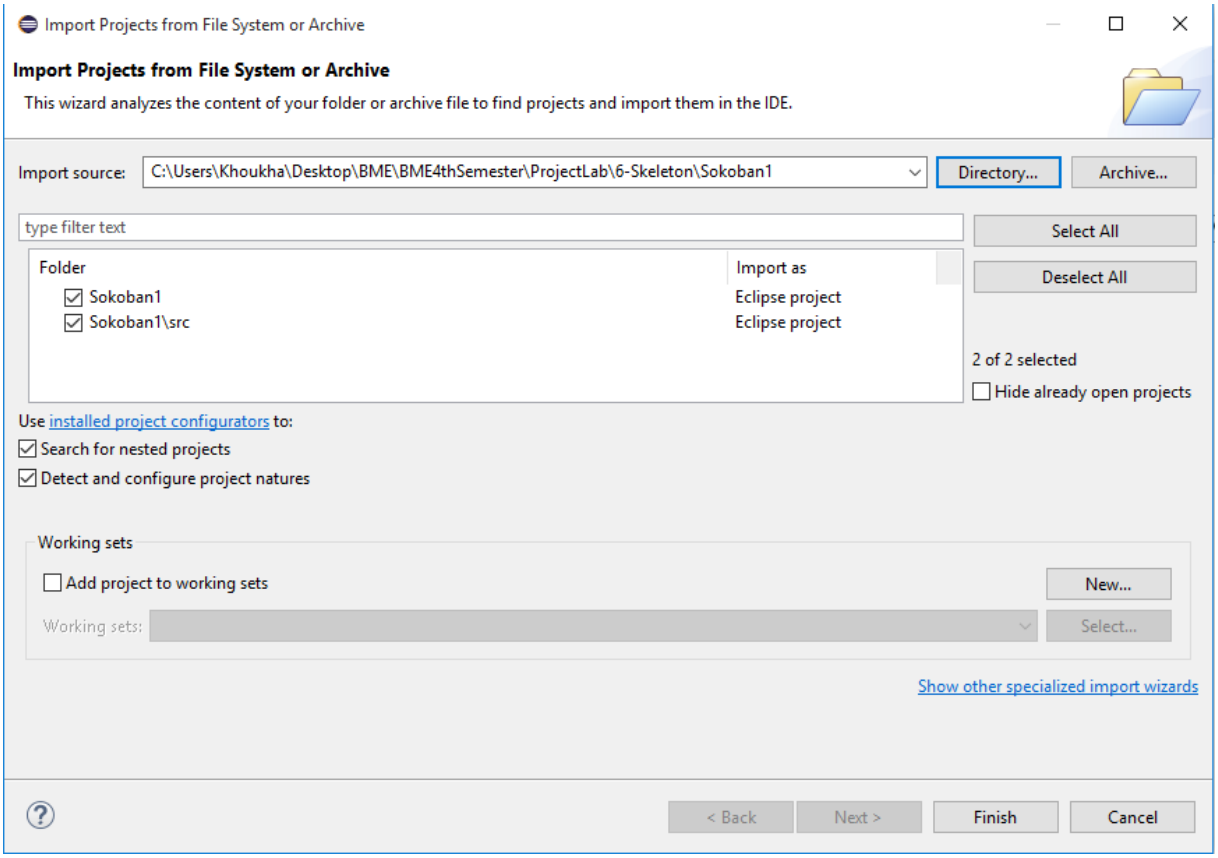
In order to compile the project the Eclipse IDE is needed, for further information about installation you can check the following official website https://www.eclipse.org/downloads/ from where you can download any version compatible with your operating system.
We would recommend installing the latest version of Eclipse IDE.

Now that everything is installed on your computer, download the zip file called Sokoban, and extract it (click on the right button of the mouse and you will find extracting options) in any directory of your choice.

After that, launch the Eclipse IDE, go to the menu tab, and in the top left corner you will find File, click on it, then look for Open Projects from File System, the following window will pop up:



Click on Directory and browse your file system to the directory you extracted the zip file in and click on it, and then click on ok.
The importation window should look like this:



Now click on Finish at the bottom and you are ready to go.
The next step is the following, go to the tool bar which should like the following:

And click on the arrow of the highlighted run button and select Main as startup project.

10.1.3 Run

After succesfully applying the steps mentionned in 10.1.2, the Main will run and the Application's console should look like the following:

```
<terminated> Main (5) [Java Application] C:\Program Files\Java\jre1.8.0_151
select 1 for loading test cases
select 2 for create custom map and move in that
2
```

Choose a number between 1 and 2 and hit the Enter button of your keyboard.

10.2 Test protocols

10.2.1 Worker Worker

Name of the tester	Houssam Mehdi
Date & time of test	19/4/2018 9pm

10.2.2 Worker Box

Name of the tester	Houssam Mehdi
Date & time of test	19/4/2018 9:05pm

10.2.3 Worker EmptySquare

Name of the tester	Houssam Mehdi
Date & time of test	19/4/2018 9:10pm

10.2.4 Worker Switch

Name of the tester	Houssam Mehdi
Date & time of test	19/4/2018 9:20pm

10.2.5 Worker DesignatedSquare

Name of the tester	Houssam Mehdi
Date & time of test	19/4/2018 9:25pm

10.2.6 Worker Hole

Name of the tester	Mahdi karray
Date & time of test	22/4/2018 11:00am
Result (failure)	load scenario 6 Scenario 6 loaded game state Game size 10 class Worker [1][1] class Hole [1][2] class Hole [1][2] Player 0 points: 0 move worker right Move executed (right) game state Game size 10 class Hole [1][2] class Hole [1][2] Player 0 points: 0 move worker right Move executed (right)
Possible causes	When a Worker is stepping on a Hole, he should disappear and be null but he doesn't because I can move him again.

Correction	In the die function(Worker class):
	public void Die() {
	
	Fields f0= this.GetField();
	f0.Remove(this);
	f0.setFree(true);
	this.player=null;
	}
	Correction:
	public void Die()
	{
	Fields f0= this.GetField();

	f0.Remove(this);
	f0.setFree(true);
	this.player.SetWorker(null);
	}

10.2.7 Worker Wall

Name of the tester	Houssam Mehdi
Date & time of test	19/4/2018 9:05pm

10.2.8 Box Worker

Name of the tester	Houssam Mehdi
Date & time of test	19/4/2018 9:05pm

10.2.1 Box Box

Name of the tester	Houssam Mehdi
Date & time of test	19/4/2018 9:05pm

10.2.2 Box EmptySquare

Name of the tester	Houssam Mehdi
Date & time of test	19/4/2018 9:05pm

10.2.3 Box Switch

Name of the tester	Houssam Mehdi	
Date & time of test	19/4/2018 12:00pm	
Result (failure)	load scenario 11 Scenario 11 loaded game state Game size 10 class Worker [1][1] class Box [1][2] class Switch [1][3] class Hole [2][2] Hole state: true class Hole [2][2] Hole state: true Player 0 points: 0 move worker right Move executed (right) game state Game size 10 class Worker [1][2] class Switch [1][3] class Box [1][3] class Hole [2][2] Hole state: false class Hole [2][2] Hole state: false Player 0 points: 0	move worker down Move executed (down) game state Game size 10 class Switch [1][3] class Box [1][3] class Hole [2][2] Hole state: true class Hole [2][2] Hole state: true Player 0 points: 0

Possible causes	When a Worker is stepping on a Hole, he's changing it's state – HitByWorker -> Hole class.

Correction	`public boolean HitByWorker(Worker w, Direction d) {`
	`if(this.state=true){`
	`Correction:`
	`public boolean HitByWorker(Worker w, Direction d) {`
	`if(this.state==true){`
	`In the HitByWorker function (Hole class), we are setting the state to true instead of comparing it.`

10.2.4 Box Switch

Name of the tester	Houssam Mehdi,Mahdi karray	
Date & time of test	19/4/2018 12:00pm	

Result (failure)	`load scenario 11`
	`Scenario 11 loaded`
	`game state`
	`Game size 10`
	`class Worker [1][1]`
	`class Box [1][2]`
	`class Switch [1][3]`
	`class Hole [2][2]`
	`Hole state: true`
	`class Hole [2][2]`
	`Hole state: true`
	`Player 0 points: 0`
	`move worker right`
	`Move executed (right)`
	`game state`
	`Game size 10`
	`class Worker [1][2]`
	`class Switch [1][3]`
	`class Box [1][3]`
	`class Hole [2][2]`
	`Hole state: false`
	`class Hole [2][2]`
	`Hole state: false`
	`Player 0 points: 0`
		`move worker down`
		`Move executed (down)`
		`game state`
		`Game size 10`
		`class Worker [1][2]`
		`class Switch [1][3]`
		`class Box [1][3]`
		`class Hole [2][2]`
		`Hole state: false`
		`class Hole [2][2]`
		`Hole state: false`
		`Player 0 points: 0`

Possible causes	When a Worker tries to step on a Hole who's state is false, which means it acts as a normal field, the move doesn't occur because we have 2 Holes on the field.	

Correction	`SetState function (Switch class)`
	`public void setState(boolean state) {`
	`this.state = state;`
	`if (state==true )`
	`{this.hole.SetHoleState(false);}`
	`else`
	`{this.hole.SetHoleState(true);`
	
	`this.hole.field.things[1]=this.hole.field.things[0];`
	`}`
	`// HitByWorker function(Hole class)`
	`public boolean HitByWorker(Worker w, Direction d) {`
	`if(this.state==true)`
	`{ //if the state of the Hole is ON, then it kills the thing`	

	w.Die();
	}
	else
	{
	Fields f0= w.GetField();// this will return the current field
	Fields f1 = f0.GetNeighbor(d);
	f0.Remove(w);
	f1.Accept(w);
	w.SetField(f1);
	f1.setFree(false);
	this.previous = w;
	}
	return true;
	}

10.2.5 Box DesignatedSquare

Name of the tester	Houssam Mehdi	
Date & time of test	19/4/2018 12:00pm	
Result (failure)	load scenario 12 Scenario 12 loaded move worker right Move executed (right) game state Game size 10 class Worker [1][2] class DesignatedSquare [1][3] class Box [1][3] Player 0 points: 5	move worker right Exception in thread "main" java.lang.ClassCastException: Worker cannot be cast to Box
Possible causes	When requesting the previous pusher of a Worker who hasn't been pushed, we're getting a cast problem.	

Correction	**Box bo = this.GetPusher(w);**
	Worker wo = (Worker)bo.previous;
	wo.GetPlayer().subtractPoints(5);
	return true;
	
	Correction:
	
	try {
	
	bo = **this**.GetPusher(w);
	
	}**catch(ClassCastException e)** {
	
	Worker wo = w;
	wo.GetPlayer().subtractPoints(5);
	return true;
	
	}
	
	Worker wo = (Worker)bo.previous;
	wo.GetPlayer().subtractPoints(5);
	return true;
	
	}	

	Surrounded it by a try catch, if the cast fails, it means it was null, therefore we set the worker to the worker itself instead of setting it to the previous pusher of the Box.

10.2.6 Box Hole

Name of the tester	Houssam Mehdi
Date & time of test	19/4/2018 12:00pm

10.2.7 sBox

Name of the tester	Houssam Mehdi
Date & time of test	19/4/2018 9:05p

10.2.8 Worker Box

Name of the tester	Houssam Mehdi
Date & time of test	19/4/2018 9:05pm

10.2.9 Worker Box (Switch+Box)

Name of the tester	Houssam Mehdi	
Date & time of test	19/4/2018 12:00pm	
Result (failure)	load scenario 15 Scenario 15 loaded game state Game size 10 class Worker [1][1] class Box [1][2] class Switch [1][3] class Box [1][3] class Hole [2][2] Player 0 points: 0	move worker right Move executed (right) game state Game size 10 class Worker [1][2] class Box [1][3] class Box [1][4] class Box [1][4] class Hole [2][2] Player 0 points: 0
Possible causes	Switch disappearing, same Box on 2 fields. Problem in the Accept or Remove.	

Correction	else if (things[1]==t)
	{
	things[0]=null;
	sizeOfThing--;
	}
	Correction:
	else if (things[1]==t)
	{
	things[1]=null;
	sizeOfThing--;
	}
	
	In the Remove function (Fields class), comparing things[1] and then making things[0] null instead of things[1].	

10.2.10 Worker Box Box Worker Box DesignatedSquare

Name of the tester	Houssam Mehdi	
Date & time of test	19/4/2018 8pm	
Result (failure)	load scenario 17 Scenario 17 loaded game state Game size 10 class Worker [1][1] class Box [1][2] class Box [1][3] class Worker [1][4] class Box [1][5] class DesignatedSquare [1][6] Player 0 points: 0 Player 1 points: 0 move worker right Move executed (right)	move worker right Move executed (right) game state Game size 10 class Worker [1][2] class Box [1][3] class Box [1][4] class Worker [1][5] class DesignatedSquare [1][6] class Box [1][6] Player 0 points: 0 Player 1 points: 0
Possible causes	Not entering the if loop in the HitByWorker – Box class	
Correction	else if((checkNull==true)&(f1.getSizeOfThings()==0)) else if((checkNull==true)&(f1.getSizeOfThings()==1)) Correction: else if((checkNull==true)&&(f2.getSizeOfThings()==0)) else if((checkNull==true)&&(f2.getSizeOfThings()==1))	

10.3 Evaluation

[Evaluation shows every team member's participation rate in the project (percents) ***from the beginning of the project*** *to the date of evaluation. The sum of percent values must be 100.]*

Name of the team member	Participation (%)
Houssam Mehdi	20
Mahdi Karray	20
Rovshan Sirinli	20
Khouloud khezami	20
Tony Azar	20

10.4 Protocol

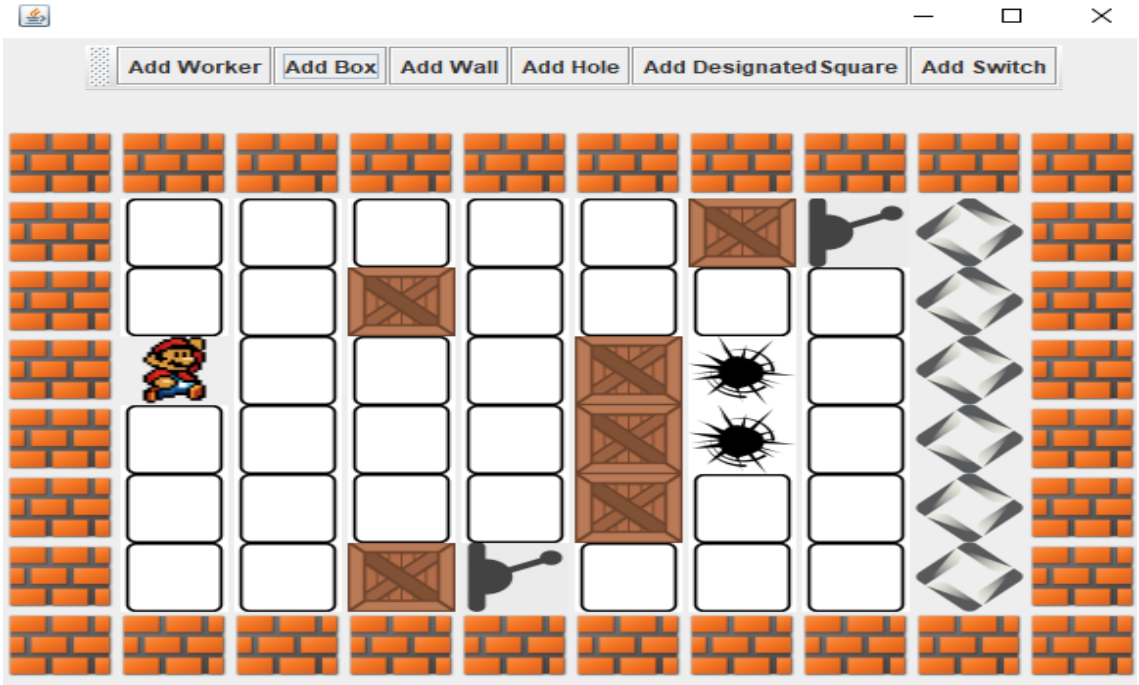
Start (date & time)	Duration (hours)	Performer(s) name	Activity description
19/04/2018	8 Hours	Team members	Meeting
21/04/2018	4 Hours	Team members	Meeting
22/04/2018	4 Hours	Team members	Meeting

11. User interface specification

11.1 Graphical User Interface

This is the main display of the game.

If the user chooses „load existing map" this would load a custom map that has entities on its fields.
Else the user creates his own personalized map, and this would look like the following



The users selects the Fields and selects an option from the displayed above (Add Worker, Add Box ect..) to add it on the field.

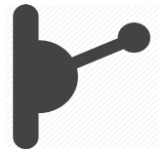
 *this icon represents the Wall class*

 *this icon represents the DesignatedSquare class*

 *this icon represents the worker class*

 *this icon represents the Hole class*

 *this icon represents the Box class*

 *this icon represents the Switch class*

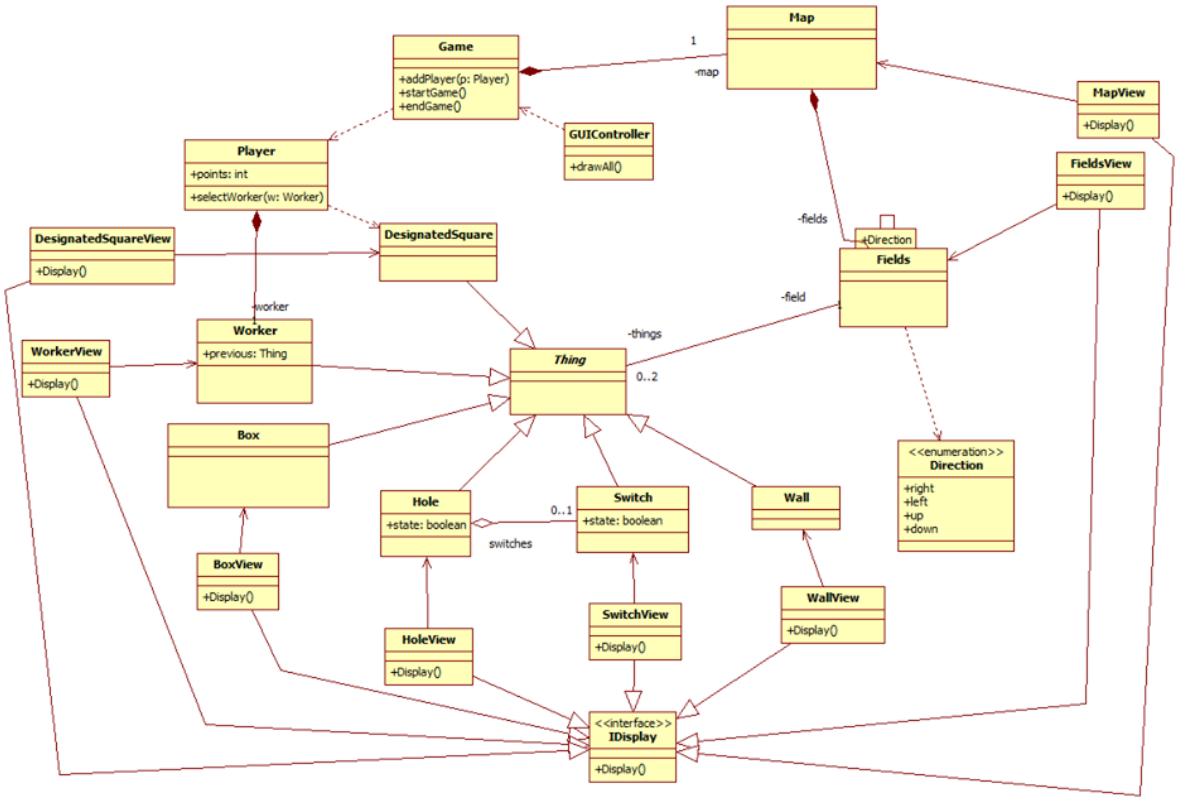
 *this icon represents the Fields class*

11.2 Architecture of the graphical system

11.2.1 Principles of the GUI

Our model is made with respect to the single responsability principal, each class that would apprear on the game's grid has an associated View class that handles the responsability of display displaying it.
All the View classes implement the same interface IDisplay.
The Icontroller takes handles the views uniformally and display them.

11.2.2 GUI Structure diagram



11.3 GUI Classes

11.3.1 GUIController

- **Responsibility**

This class handles the updating of the grid according to the user input direction

- **Superclasses**
  - -
- **Interfaces**
  - -
- **Attributes**
  - -
- **Methods**
  - *Public void DrawAll() : This method will call the Display() method of the map attribute of the Game class.*

11.3.2 MapView

- **Responsibility**

  Display the elements on the Map

- **Superclasses**
  -
- **Interfaces**
  IDispalay
- **Attributes**
  -
- **Methods**
  Public void Display():go through all the Idisplay entities and call their Display() methods respectively.

11.3.3 WallView

- **Responsibility**
  Display the Wall class
- **Superclasses**
  -
- **Interfaces**
  IDispalay
- **Attributes**
  -
- **Methods**
  Public void Display(): this method draws a Wall on the designated Field when created

11.3.4 HoleView

- **Responsibility**
  Display the Hole class
- **Superclasses**
  -
- **Interfaces**
  IDispalay
- **Attributes**
  -
- **Methods**
  Public void Display(): this method draws a Hole on the designated Field when created

11.3.5 DesignatedSquareView

- **Responsibility**
  Display the DesignatedSquare class
- **Superclasses**
  -
- **Interfaces**
  IDispalay
- **Attributes**
  -
- **Methods**
  Public void Display(): this method draws a DesignatedSquare on the designated Field when created

11.3.6 SwitchView

- **Responsibility**
  Display the Switch class
- **Superclasses**
  -
- **Interfaces**
  IDispalay
- **Attributes**
  -
- **Methods**
  Public void Display(): this method draws a Switch on the designated Field when created

11.3.7 FieldsView

- **Responsibility**
  Display the Fields class
- **Superclasses**
  -

- **Interfaces**
  IDispalay
- **Attributes**
  -
- **Methods**
  Public void Display(): this method draws a Fields when created

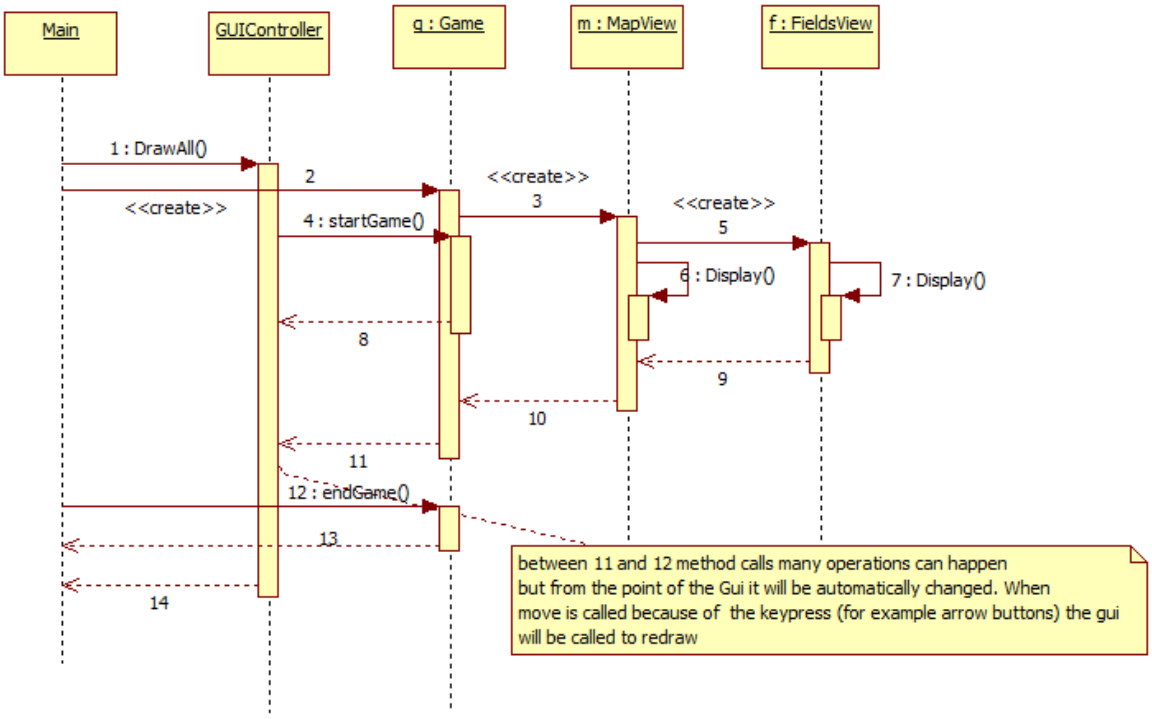
11.3.8 BoxView

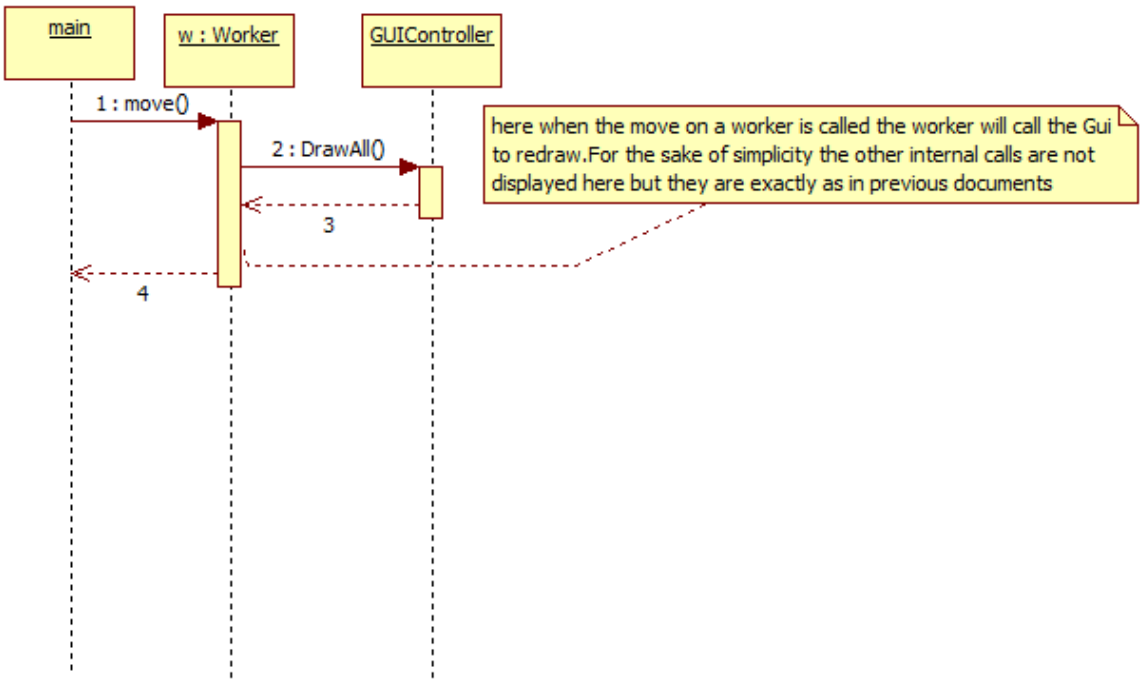
- **Responsibility**
  Display the Box class
- **Superclasses**
  -
- **Interfaces**
  IDispalay
- **Attributes**
  -
- **Methods**
  Public void Display(): this method draws a Box on the designated Field when created or moved.

11.3.9 WorkerView

- **Responsibility**
  Display the Worker class
- **Superclasses**
  -
- **Interfaces**
  IDispalay
- **Attributes**
  -
- **Methods**
  Public void Display(): this method draws a Worker on the designated Field when created or moved

11.4 Dynamic connection between the model and the GUI





11.5 Protocol

Start (date & time)	Duration (hours)	Performer(s) name	Activity description
01/05/2018	*5hours*	*Team's members*	*Meeting*

13. Complete program

13.1.1 List of files

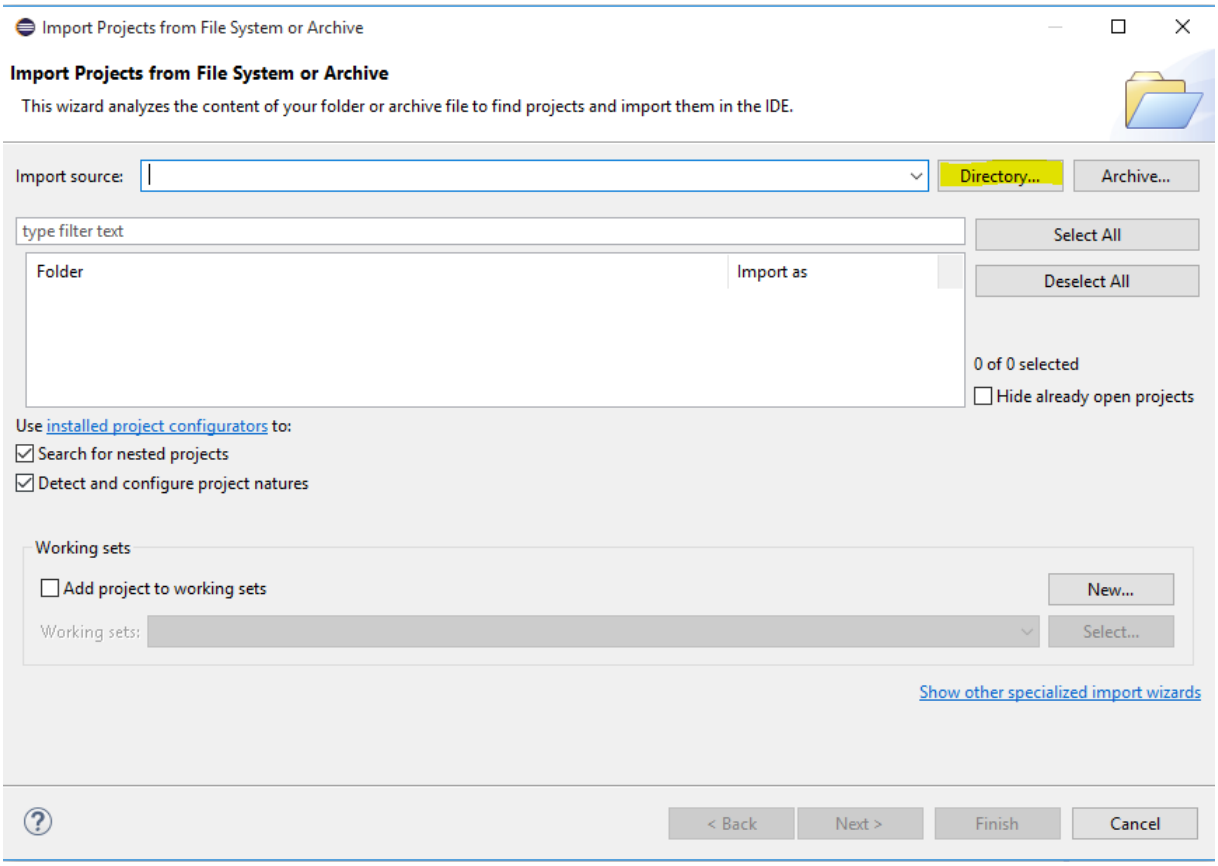
File name	Size	Date	Content
Box.java	4kb	4/23/2018 2:37AM	Box class
Worker.java	4kb	04/23/2018 2:37AM	Worker class
DesignatedSquare.java	3kb	4/23/2018 2:37AM	DS class
Direction.java	1kb	4/23/2018 2:37AM	Enum Directions
Fields.java	3kb	4/23/2018 2:37AM	Fields class
Game.java	9kb	5/13/2018 7:31PM	Game class
Hole.java	1kb	5/13/2018 7:31PM	Hole class
Main.java	18kb	4/23/2018 2:37AM	Main with all the tests
Map.java	2kb	4/19/2018 2:37AM	Map class
Player.java	2kb	4/19/2018 2:37AM	Player class
Switch.java	1kb	5/13/2018 7:31PM	Switch class
Thing.java	1kb	4/19/2018 2:37AM	Thing class
Wall.java	1kb	4/19/2018 2:37AM	Wall class
JPlay	36kb	5/13/2018 7:31PM	Play class
JBuilder	202kb	5/13/2018 7:31PM	Builder class
JMenu	4kb	5/13/2018 7:31PM	Menu class

13.1.2 Compilation

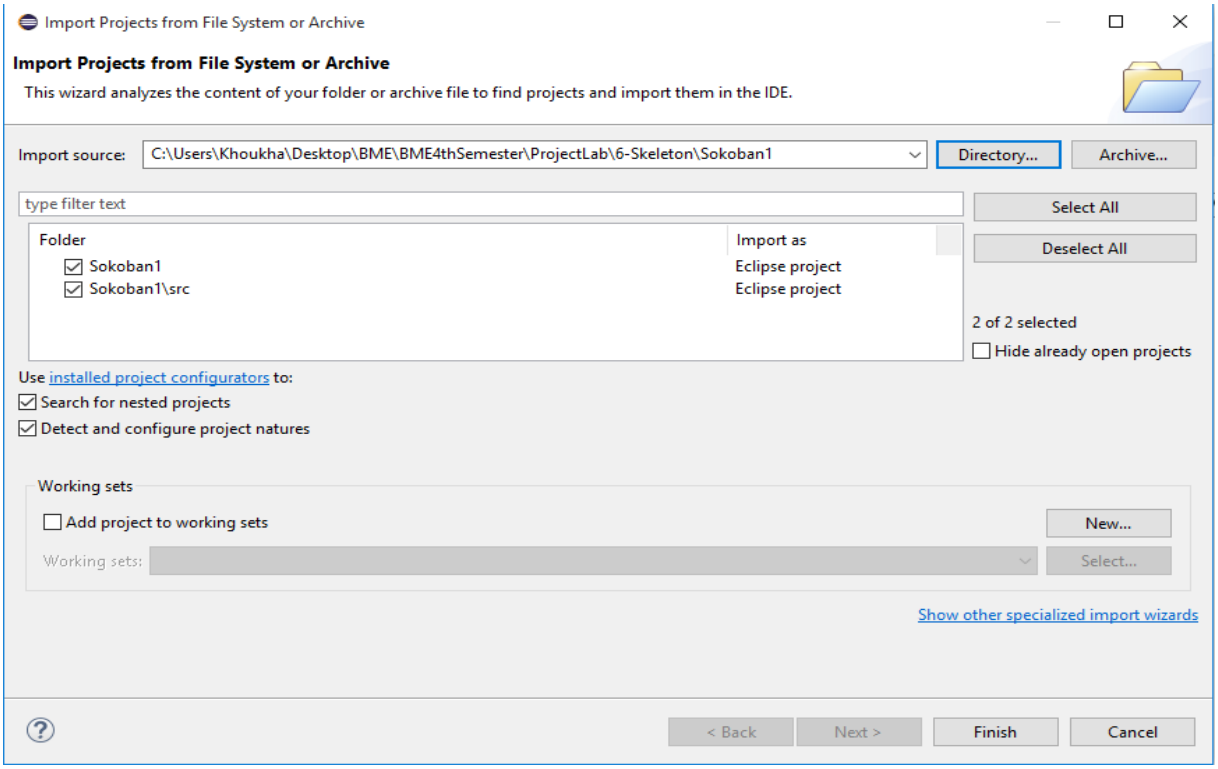
In order to compile the project the Eclipse IDE is needed, for further information about installation you can check the following official website https://www.eclipse.org/downloads/ from where you can download any version compatible with your operating system.
We would recommend installing the latest version of Eclipse IDE.

Now that everything is installed on your computer, download the zip file called Sokoban, and extract it (click on the right button of the mouse and you will find extracting options) in any directory of your choice.

After that, launch the Eclipse IDE, go to the menu tab, and in the top left corner you will find File, click on it, then look for Open Projects from File System, the following window will pop up:



Click on Directory and browse your file system to the directory you extracted the zip file in and click on it, and then click on ok.
The importation window should look like this:



Now click on Finish at the bottom and you are ready to go.
The next step is the following, go to the tool bar which should like the following:

And click on the arrow of the highlighted run button and select Main as startup project.

13.1.3 Run

After succesfully applying the steps mentionned in 13.1.2, the JMenu will run and the Application should look like the following:

13.2 Evaluation

Name of the team member	Participation (%)
Rovshan Shirinli	20
Houssam Mehdi	20
Khouloud Khezami	20
M'ahdi Karray	20
Tony Azar	20

13.3 Protocol

Start (date & time)	**Duration (hours)**	**Performer(s) name**	**Activity description**
5/06/2018 10:00 AM	*8 Hours*	*Member's Team*	*Swing*
5/13/2018	2 Hours	Member's Team	Testing some cases