for each: foreach
do
end foreach

Algorithm 1: Overlapping-Event-Scheduler

Input : EventList : List< *Event* >, NewEvent : Event
Output: ValidEventsFlag : Boolean, OverlappingEvent:Event

```
1  foreach e1 ∈ EventList do
2  |  dummyst1 = e1.startTime
3  |  dummyst2 = NewEvent.startTime
4  |  if e1.chosenMobility! = NULL then
5  |  |  dummyst1 = dummyst1 - e1.chosenMobility.TravelDuration
6  |  end
7  |  if NewEvent.chosenMobility! = NULL then
8  |  |  dummyst2 = dummyst - NewEvent.chosenMobility.TravelDuration
9  |  end
10 |  // Checks overlapping
11 |  if e1.startTime < NewEvent.endTime and e1.endTime >
   |   NewEvent.startTime then
12 |  |  //Overlap occurs
13 |  |  ValidEventsFlag = False
14 |  |  return ValidEventsFlag,e1
15 |  else
16 |  |  ValidEventsFlag = True
17 |  |  return ValidEventsFlag,NULL
18 |  end
19 end
20
```

Algorithm 2: Mobility-Option-Recommender-For-Events

Input : PreferenceList : List< *Mobility* >,NewEvent:*Event*,EventList: List< *Event* >,EmptyList: List< *Break* >

Output: RecommendedMobilityList : List< *Mobility* >

```
1  EventList.push(NewEvent) RecommendedMobilityList = ∅
2  foreach m ∈ PreferenceList do
3  |   dummye = new Event
4  |   dummye.startTime = NewEvent.startTime - m.TravelDuration
5  |   dummye.endTime = NewEvent.startTime
6  |   dummye.chosenMobility = NULL
7  |   foreach empty in EmptyList do
8  |   |   // If mobility m is in a empty interval
9  |   |   if dummye.startTime > empty.startTime and dummye.endTime
   |   |    < empty.endTime then
10 |   |   |   //Add m to recommendation list
11 |   |   |   RecommendedMobilityList.push(m)
12 |   |   end
13 |   end
14 end
15 return RecommendedMobilityList
16
```

Algorithm 3: Empty-Slot-Generator

Input : EventList : List $< Event >$
Output: EmptySlotList : List $< Break >$

```
1  // Initialization
2  EmptySlotList =
     Break(startTime = 0, endTime = 24.00, chosenMobility = None)
3  dummye = new < Break >
4  dummyst = new < DateTime >
5  foreach e1 ∈ EventList do
6      foreach e2 ∈ EmptySlotList do
7          if e2.startTime < e1.startTime and e1.endTime < e2.endTime
             then
8              if e1! = Break then
9                  dummyst = e1.startTime -
                     e1.chosenMobility.travelDuration
10             end
11             // Partition of the empty slots as two new events
12             EmptyList.delete(e2)
13             dummye.startTime = e2.startTime
14             dummye.endTime = dummyst
15             dummye.Duration = dummye.endTime - dummye.startTime
16             EmptyList.push(dummye)
17             dummye.startTime = e1.endTime
18             dummye.endTime = e2.endTime
19             dummye.Duration = dummye.endTime - dummye.startTime
20             EmptyList.push(dummye)
21         end
22     end
23 end
```

Algorithm 4: Locator-For-Breaks

Input : BreakList:List< *Break* >, EmptyList : List< *Break* >
Output: ValidScheduleWithBreaks:Boolean , newBreakList :List< *Break* >,EmptyList : List< *Break* >

```
1  newBreakList = List< Break > foreach Break ∈ BreakList do
2      foreach empty in EmptyList do
3          if Break.startTime < empty.endTime and Break.endTime >
             empty.startTime then
4              dummyst = max(Break.startTime,empty.startTime)
5              dummyend = min(Break.endTime,empty.endTime)
6              AvailDuration = dummyend - dummyst
7              if AvailDuration > New-
                 Break.Duration+NewBreak.chosenMobility.TravelDuration
                 then
8                  empty.Duration = empty.Duration - (New-
                     Break.Duration+NewBreak.chosenMobility.TravelDuration)
                   newBreakList.push(Break)
9              end
10         end
11     end
12 end
13 // If all the breaks are schedulable, schedule is valid with breaks
14  ValidScheduleWithBreaks = IsSame(newBreakList, BreakList)
15 return ValidScheduleWithBreaks,newBreakList,EmptyList
16
```

Algorithm 5: Mobility-Option-Recommender-For-Breaks

Input : PreferenceList,EventList : List $< Event >$,
NewBreak$< Break >$,EmptyList : List$< Break >$
Output: RecommendedMobilityList: List$< Mobility >$

```
1  RecommendedMobilityList = ∅
2  foreach empty in EmptyList do
3  |  // If empty slot and new break overlaps
4  |  if Break.startTime < empty.endTime and Break.endTime >
   |   empty.startTime then
5  |  |  // Calculate the available duration
6  |  |  dummyst = max(Break.startTime,empty.startTime)
7  |  |  dummyend = min(Break.endTime,empty.endTime)
8  |  |  AvailDuration = dummyend - dummyst
9  |  |  foreach m in PreferenceList do
10 |  |  |  if AvailDuration > NewBreak.Duration+m.TravelDuration
   |  |  |   then
11 |  |  |  |  RecommendedMobilityList.push(m)
12 |  |  |  end
13 |  |  end
14 |  end
15 end
16 return RecommendedMobilityList
```

Algorithm 6: AddEvent

Input : PL:List $< Mobility >$,EventList: List$< Event >$, BreakList: List$< Break >$, sT:DateTime,eT:DateTime,EL:Location

Output: EventList: $< Event >$

```
// This piece of algorithm chart explains
// overall flow of the algorithms and user
// interaction
newEvent = new
  Event(startTime = sT, endTime = eT, eventLocation = EL)
ValidEventsFlag,OverlapEvents =
  Overlapping-Event-Scheduler(EventList,newEvent)
if ValidEventsFlag == True then
    EmptyList = Empty-List-Generator(EventList)
    ValidScheduleWithBreaks , newBreakList ,EmptyList =
      Locator-For-Breaks(BreakList,EmptyList)
    if ValidScheduleWithBreaks = True then
        RecommendedMobL = Mobility-Option-Recommender-For-
          Events(PL,newEvent,EventList,EmptyList)
        if isEmpty(RecommendedMobL) then
            return UnreachableError
        else
            EventList.push(newEvent)
            return EventList
        end
    else
        return OverlapError
    end
else
    return OverlapError
end
```

Algorithm 7: AddBreak

Input : PL:List $< Mobility >$,EventList: List$< Event >$, BreakList: List$< Break >$, sT:DateTime,eT:DateTime,duration:DateTime,EL:Location

Output: EventList: $< Event >$

```
1  // This piece of algorithm chart explains
2  // overall flow of the algorithms and user
3  // interaction
4  newBreak = new Break(startTime = sT, endTime = eT, Duration =
     duration, eventLocation = EL)
5  EmptyList = Empty-List-Generator(EventList)
6  ValidScheduleWithBreaks , newBreakList ,EmptyList =
     Locator-For-Breaks(BreakList,EmptyList)
7  if ValidScheduleWithBreaks = True then
8  |   RecommendedMobL = Mobility-Option-Recommender-For-
   |     Breaks(PL,newEvent,EventList,EmptyList)
9  |   if isEmpty(RecommendedMobL) then
10 |   |   return UnreachableError
11 |   else
12 |   |   BreakList.push(Break)
13 |   |   return BreakList
14 |   end
15 else
16 |   return OverlapError
17 end
```