7.4 GITTER-GAMES, SOLITAIRE BRETTSPIEL

EINFÜHRUNG

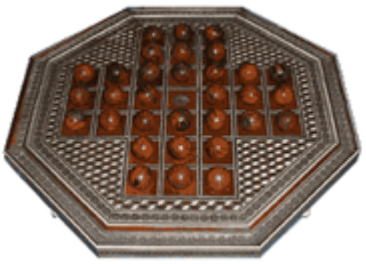
Bei einer bestimmten Klasse von Computergames können sich die Spielfiguren nur in Zellen eine Gitterstruktur aufhalten, wobei die Zellen meist gleiche Grösse haben und matrixartig angeordnet sind. Di Berücksichtigung der Ortsbeschränkung auf eine Gitterstruktur vereinfacht die Implementierung des Spie ganz wesentlich. Wie schon der Name sagt, ist die Gamelibrary *JGameGrid* für gitterartige Game besonders optimiert.

In diesem Kapitel entwickelst du schrittweise das Brett-Solitaire mit dem englischen Brettlayout. Dab lernst du wichtige Lösungsverfahren kennen, die du auf alle Gitterspiele anwenden kannst.

PROGRAMMIERKONZEPTE: *Spielbrett, Spielregeln, Pflichtenheft, Game-Over*

BRETTINITIALISIERUNG, MAUSSTEUERUNG

Auf einem Spielbrett befinden sich in einer regelmässigen Anordnung Löcher oder Vertiefungen, in die du Stifte stecken bzw. Murmeln legen kannst. Das bekannteste Board-Solitaire verwendet ein Brett mit einer kreuzartigen Anordnung von 33 Vertiefungen und wird englisches Brett genannt. Am Anfang sind alle Löcher ausser dem Zentrumsloch mit Murmeln belegt. Wie der Name *Solitaire* sagt, wird es meistens von einer einzigen Person gespielt.



Ein englisches Board-Solitair aus Indien, 1830

Es gelten folgende **Spielregeln**: Ein Zug besteht darin, eine Murmel auf ein freies Loch zu verschieben, wobei dabei genau eine Murmel in horizontaler oder vertikaler Richtung übersprungen werden muss. Die übersprungene Murmel wird vom Spielbrett entfernt.

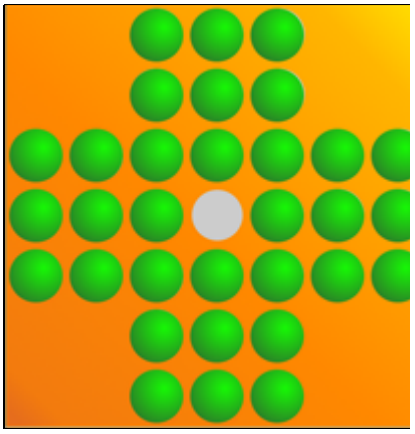
Das Ziel besteht darin, alle Murmeln bis auf eine letzte vom Brett "abzuräumen". Das Spiel gilt a besonders gut gelöst, falls sich die letzte Murmel im Zentrum befindet. In der Implementierung a Computergame sollst du eine bestimmte Murmel durch Drücken der Maustaste "packen" und b gedrückter Maustaste verschieben können. Beim Loslassen der Maustaste wird geprüft, ob der Zug de Spielregeln entspricht. Falls er regelwidrig ist, soll die Murmel wieder an den Anfangsort zurückspringen; i er legal, so wird die Murmel am neuen Ort angezeigt und die übersprungene Murmel vom Brett entfernt.

Damit ist das **Pflichtenheft** klar und du kannst hinter die Implementierung gehen. Diese erfolgt wie imme schrittweise, wobei bei jedem Schritt ein lauffähiges Programm vorliegen muss. Es liegt auf der Hand, ein GameGrid mit 7x7 Zellen zu verwenden, wobei die Eckzellen nicht verwendet werden. Zuerst zeichnest d in der Funktion *initBoard()* mit dem Hintergrundbild *solitaire_board.png*, die sich in der Distribution vo TigerJython befindet, das Brett und realisierst die Maussteuerung mit den Maus-Callbacks *mousePresse mouseDragged* und *mouseReleased*.

Beim Press-Event merkst du dir die aktuelle Location und die aktuelle Murmel in dieser Zelle. Diese wird d mit getOneActorAt() zurückgeben, wobei du *None* erhältst, wenn die Zelle leer ist. Der Entwicklungsproze ist leichter zu beherrschen und Fehler besser aufzufinden, wenn du in einer Statusbar (oder in der Konsol wichtige Ergebnisse ausschreibst.

Beim Drag-Event verschiebst du das sichtbare Bild der Murmel mit setLocationOffset() an die aktuelle Cursorposition. Diese kann auch neben der Zellenmitten liegen, sodass sich eine kontinuierlich sichtbare Verschiebung ergibt. Dabei ist es wichtig, dass du den Murmelactor selbst nicht verschiebst, sondern nur sein Spritebild (darum die Bezeichnung *Offset*). Damit umgehst du alle Schwierigkeiten mit übereinander liegenden Actors.

Beim Release-Event soll in dieser ersten Version die Kugel wieder an ihre Startlocation zurückspringen. Dies erreichst du mit dem Aufruf von setLocationOffset(0, 0).



```
from gamegrid import *

def isMarbleLocation(loc):
    if loc.x < 0 or loc.x > 6 or loc.y < 0 or loc.y > 6:
        return False
    if loc.x in [0, 1, 5, 6] and loc.y in [0, 1, 5, 6]:
        return False
    return True

def initBoard():
    for x in range(7):
        for y in range(7):
            loc = Location(x, y)
            if isMarbleLocation(loc):
                marble = Actor("sprites/marble.png")
                addActor(marble, loc)
    removeActorsAt(Location(3, 3)) # Remove marble in center

def pressEvent(e):
    global startLoc, movingMarble
    startLoc = toLocationInGrid(e.getX(), e.getY())
    movingMarble = getOneActorAt(startLoc)
    if movingMarble == None:
       setStatusText("Pressed at " + str(startLoc) + ". No marble found")
    else:
       setStatusText("Pressed at " + str(startLoc) + ". Marble found")

def dragEvent(e):
    if movingMarble == None:
        return
    startPoint = toPoint(startLoc)
    movingMarble.setLocationOffset(e.getX() - startPoint.x,
                                   e.getY() - startPoint.y)

def releaseEvent(e):
    if movingMarble == None:
        return
    movingMarble.setLocationOffset(0, 0)

makeGameGrid(7, 7, 70, None, "sprites/solitaire_board.png", False,
    mousePressed = pressEvent, mouseDragged = dragEvent,
    mouseReleased = releaseEvent)
setBgColor(Color(255, 166, 0))
setSimulationPeriod(20)
addStatusBar(30)
setStatusText("Press-drag-release to make a move.")
initBoard()
```

```
show()
doRun()
```

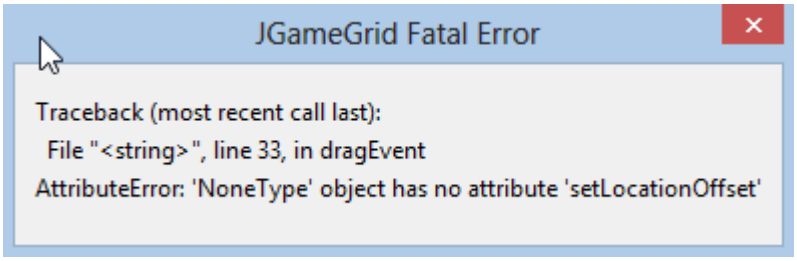
Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

MEMO

Statt beim Draggen den Actor selbst zu verschieben, kannst du auch nur sein Spritebild bewegen. Daz verwendest du setLocationOffset(x, y), wobei x und y relative Koordinaten in Bezug auf den aktuelle Mittelpunkt des Actors sind.

Im Zusammenhang mit Mausbewegungen musst du zwischen den Koordinaten der Maus un Zellenkoordinaten sorgfältig unterscheiden. Dabei sind die Konversionsfunktione *toLocationInGrid(pixel_coord)* bzw. *toPoint(location_coord)* wichtig.

Gehst du von einer leeren Zelle aus, so führen der *Drag-* und *Release-Event* zu einem berüchtigte Programm-Absturz, da *movingMarble* den Wert *None* hat und du damit eine Methode aufrufst.



Um den Fehler zu vermeiden, verlässt du die Callbacks gerade zu Beginn mit einem sofortigen return.

IMPLEMENTIERUNG DER SPIELREGELN

Wie würdest du beim wirklichen Spiel die Spielregeln überprüfen? Du müsstest wissen, mit welcher Murm du gestartet bist, also deren Startlocation *start* kennen. Dann müsstest du wissen, wohin du die Murm verschieben möchtest, also deren Ziellocation *dest* kennen. Für einen legalen Zug müssen folgende Bedingungen zutreffen:

1. Bei *start* gibt es eine Murmel
2. Bei *dest* gibt es keine Murmel
3. *dest* ist eine Zelle, die zum Board gehört
4. *start* und *dest* sind entweder horizontal oder vertikal zwei Zellen auseinander
5. An der Zwischenzelle befindet sich eine Murmel

Es ist elegant, diese Bedingungen in einer Funktion getRemoveMarble(start, dest) zu implementiere welche bei einem legalen Zug die zu entfernende Murmel und bei einem illegalen Zug *None* zurückgibt.

Es ist klar, dass du diese Funktion im Release-Event aufrufst und bei einem legalen Zug de zurückgegebenen Actor mit removeActor() vom Board nimmst.

```
from gamegrid import *

def getRemoveMarble(start, dest):
    if getOneActorAt(start) == None:
        return None
    if getOneActorAt(dest) != None:
        return None
    if not isMarbleLocation(dest):
        return None
    if dest.x - start.x == 2 and dest.y == start.y:
        return getOneActorAt(Location(start.x + 1, start.y))
    if start.x - dest.x == 2 and dest.y == start.y:
        return getOneActorAt(Location(start.x - 1, start.y))
    if dest.y - start.y == 2 and dest.x == start.x:
```

```
        return getOneActorAt(Location(start.x, start.y + 1))
    if start.y - dest.y == 2 and dest.x == start.x:
        return getOneActorAt(Location(start.x, start.y - 1))

def isMarbleLocation(loc):
    if loc.x < 0 or loc.x > 6 or loc.y < 0 or loc.y > 6:
        return False
    if loc.x in [0, 1, 5, 6] and loc.y in [0, 1, 5, 6]:
        return False
    return True

def initBoard():
    for x in range(7):
        for y in range(7):
            loc = Location(x, y)
            if isMarbleLocation(loc):
                marble = Actor("sprites/marble.png")
                addActor(marble, loc)
    removeActorsAt(Location(3, 3)) # Remove marble in center

def pressEvent(e):
    global startLoc, movingMarble
    startLoc = toLocationInGrid(e.getX(), e.getY())
    movingMarble = getOneActorAt(startLoc)
    if movingMarble == None:
       setStatusText("Pressed at " + str(startLoc) + ". No marble found")
    else:
       setStatusText("Pressed at " + str(startLoc) + ". Marble found")

def dragEvent(e):
    if movingMarble == None:
        return
    startPoint = toPoint(startLoc)
    movingMarble.setLocationOffset(e.getX() - startPoint.x,
                                   e.getY() - startPoint.y)

def releaseEvent(e):
    if movingMarble == None:
        return
    destLoc = toLocationInGrid(e.getX(), e.getY())
    movingMarble.setLocationOffset(0, 0)
    removeMarble = getRemoveMarble(startLoc, destLoc)
    if removeMarble == None:
        setStatusText("Released at " + str(destLoc) + ". Not a valid move.")
    else:
        removeActor(removeMarble)
        movingMarble.setLocation(destLoc)
        setStatusText("Released at " + str(destLoc)+ ". Marble removed.")


startLoc = None
movingMarble = None

makeGameGrid(7, 7, 70, None, "sprites/solitaire_board.png", False,
    mousePressed = pressEvent, mouseDragged = dragEvent,
    mouseReleased = releaseEvent)
setBgColor(Color(255, 166, 0))
setSimulationPeriod(20)
addStatusBar(30)
setStatusText("Press-drag-release to make a move.")
initBoard()
show()
doRun()
```

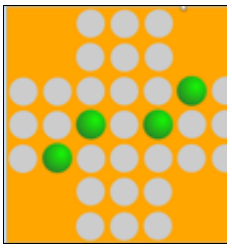
Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

MEMO

Statt mit mehreren vorzeitigen *return* die Funktion *getRemoveMarble()* zu verlassen, könnte man d Bedingungen auch mit booleschen Operation verknüpfen. Es ist Ansichtssache, welch Programmiertechnik man als geeigneter betrachtet.

PRÜFUNG AUF GAME-OVER

Es bleibt dir jetzt nur noch die Aufgabe, bei jedem legalen Zug zu prüfen, ob das Spiel beendet ist. Dies i sicher dann der Fall, falls sich nur noch eine einzige Murmel auf dem Spielfeld befindet und du damit da Spielziel erreicht hast.



Dabei vergisst du aber, dass es noch andere Spielkonstellationen geben könnte, bei denen das Spiel als beendet betrachtet werden muss, nämlich wenn sich noch mehr als eine Murmel auf dem Brett befindet, du aber mit keiner davon einen legalen Zug machen kannst. Es ist zwar nicht ganz sicher, ob man mit legalen Zügen überhaupt einmal in diese Situation kommt, aber du musst ***defensiv* programmieren**, also immer auf der sicheren Seite bleiben, denn du kannst damit rechnen, dass auch beim Programmieren der Murphy-Spruch gilt: "Wenn etwas schief gehen kann, geht es schief".

Um diese Situation in den Griff zu bekommen, kannst du in der Funktion isMovePossible() alle noc vorhandenen Murmeln einzeln darauf zu testen, ob man mit ihnen einen legalen Zug machen kann. Daz prüfst du für jede Murmel, ob es mit irgendeinem Loch eine zu entfernende Zwischenmurmel gibt [mehr...]

```
from gamegrid import *

def checkGameOver():
    global isGameOver
    marbles = getActors() # get remaining marbles
    if len(marbles) == 1:
        setStatusText("Game over. You won.")
        isGameOver = True
    else:
        # check if there are any valid moves left
        if not isMovePossible():
           setStatusText("Game over. You lost. (No valid moves available)")
           isGameOver = True

def isMovePossible():
   for a in getActors():  # run over all remaining marbles
        for x in range(7): # run over all holes
            for y in range(7):
                loc = Location(x, y)
                if getOneActorAt(loc) == None and \
                  getRemoveMarble(a.getLocation(), Location(x, y)) != None:
                    return True
   return False

def getRemoveMarble(start, dest):
    if getOneActorAt(start) == None:
        return None
    if getOneActorAt(dest) != None:
        return None
    if not isMarbleLocation(dest):
        return None
    if dest.x - start.x == 2 and dest.y == start.y:
```

```
        return getOneActorAt(Location(start.x + 1, start.y))
    if start.x - dest.x == 2 and dest.y == start.y:
        return getOneActorAt(Location(start.x - 1, start.y))
    if dest.y - start.y == 2 and dest.x == start.x:
        return getOneActorAt(Location(start.x, start.y + 1))
    if start.y - dest.y == 2 and dest.x == start.x:
        return getOneActorAt(Location(start.x, start.y - 1))
    return None

def isMarbleLocation(loc):
    if loc.x < 0 or loc.x > 6 or loc.y < 0 or loc.y > 6:
        return False
    if loc.x in [0, 1, 5, 6] and loc.y in [0, 1, 5, 6]:
        return False
    return True

def initBoard():
    for x in range(7):
        for y in range(7):
            loc = Location(x, y)
            if isMarbleLocation(loc):
                marble = Actor("sprites/marble.png")
                addActor(marble, loc)
    removeActorsAt(Location(3, 3)) # Remove marble in center

def pressEvent(e):
    global startLoc, movingMarble
    if isGameOver:
        return
    startLoc = toLocationInGrid(e.getX(), e.getY())
    movingMarble = getOneActorAt(startLoc)
    if movingMarble == None:
       setStatusText("Pressed at " + str(startLoc) + ".No marble found")
    else:
       setStatusText("Pressed at " + str(startLoc) + ".Marble found")

def dragEvent(e):
    if isGameOver:
        return
    if movingMarble == None:
        return
    startPoint = toPoint(startLoc)
    movingMarble.setLocationOffset(e.getX() - startPoint.x,
                                   e.getY() - startPoint.y)

def releaseEvent(e):
    if isGameOver:
        return
    if movingMarble == None:
        return
    destLoc = toLocationInGrid(e.getX(), e.getY())
    movingMarble.setLocationOffset(0, 0)
    removeMarble = getRemoveMarble(startLoc, destLoc)
    if removeMarble == None:
        setStatusText("Released at " + str(destLoc)
                      + ". Not a valid move.")
    else:
        removeActor(removeMarble)
        movingMarble.setLocation(destLoc)
        setStatusText("Released at " + str(destLoc)+
                      ". Valid move - Marble removed.")
        checkGameOver()
```

```
startLoc = None
movingMarble = None
isGameOver = False

makeGameGrid(7, 7, 70, None, "sprites/solitaire_board.png", False,
   mousePressed = pressEvent, mouseDragged = dragEvent,
   mouseReleased = releaseEvent)
setBgColor(Color(255, 166, 0))
setSimulationPeriod(20)
addStatusBar(30)
setStatusText("Press-drag-release to make a move.")
initBoard()
show()
doRun()
```

Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

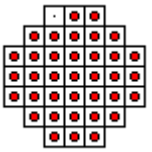
MEMO

Nach jedem Zug testest du mit checkGameOver(), ob das Spiel beendet ist. Ist dies der Fall, befindet sic das Spiel in einem ganz **speziellen Zustand,** den du mit der booleschen Variable (ein Flag) isGameOver True kennzeichnest.

Insbesondere musst du bei *Game-Over* auch alle Maus-Aktionen unterbinden. Du erreichst dies mit einer sofortigen return aus den Maus-Callbacks.

AUFGABEN

1. Erstelle ein Brett-Solitaire mit einem französischen Brett.



2. Erweitere das Brett-Solitaire mit einem Score, der die Anzahl Züge zählt und ausschreibt. Auch sc das Spiel mit der Space-Taste wieder neu gestartet werden können.

3. Orientiere dich bei einer Lehrperson oder im Internet über Lösungsstrategien des Brett-Solitair [mehr...].

4. Erstelle ein Brett-Solitaire nach deinen eigenen Ideen.