Aufgaben Protokoll

IPC-Projekt

Softwareentwicklung
5BHITT 2015/16

Maximilian Seidl

Version 0.1

Note:

Betreuer: D. Dolezal

Begonnen am 18. Februar 2017

Beendet am 18. Februar 2017

Inhaltsverzeichnis

1 Einführung

Implementiere einen einfachen Spiele-Client zum gegebenen Spiele-Server!

(Server-GUI)
Die Regeln des Spiels:

- Es spielen immer zwei Clients gegeneinander
- Das Spielbrett ist 10x10 Felder groß, wobei man über den Rand hinausgehen kann und dann wieder auf der gegenüberliegenden Seite landet
- Ziel des Spiels ist, die Schriftrolle (mit "XX" rot markiert) einzusammeln und in die gegnerische Burg (schwarzes bzw. weißes Feld) zu bringen, um sie damit zu erobern
- Spieler 1 (P1) startet auf seiner eigenen Burg (schwarz), Spieler 2 (P2) ebenfalls (weiß)
- Es gibt außer den Burgen noch folgende Spielfelder:
    - Wiese (hellgrün): Befindet man sich auf einer Wiese, sieht man zwei Felder weit, also insgesamt 5x5 Felder
    - Wald (dunkelgrün): Hier ist die Sicht eingeschränkt und man sieht nur ein Feld weit, also insgesamt 3x3 Felder
    - Berg (grau): Am Berg hat man einen guten Ausblick und sieht drei Felder weit, also insgesamt 7x7 Felder. Allerdings kostet es zwei Züge, um auf den Berg zu klettern
    - Teich (blau): Den Teich darf man nicht betreten, da man sonst das Spiel verliert.

1.1Aufgabenstellung

Erstelle einen Java- ODER Python-Client, der sich mit dem Server verbindet und selbstständige Entscheidungen trifft!

Grundanforderungen (50%):

- Nimm den simplen "manuellen" Client als Vorlage, um zu sehen, wie die Kommunikation mit dem Server funktioniert

- Dein Client-Programm wird über die Kommandozeile gestartet, wobei
  - der Port
  - die IP-Adresse des Servers
  - die Zeilenanzahl des Spielfeldes (Größe)
  - die Spaltenanzahl des Spielfeldes (Größe)
    als Kommandozeilenparameter übergeben wird
- Dein Client lässt den Benutzer einen Spielernamen wählen und überträgt diesen nach dem Verbindungsaufbau an den Server. Der Server bestätigt die Nachricht mit "OK"
- Der Client schickt anschließend basierend auf den Antworten des Servers **selbstständig** (ohne Benutzereingaben) Move-Befehle:
  - "UP": Nach oben bewegen
  - "RIGHT": Nach rechts bewegen
  - "DOWN": Nach unten bewegen
  - "LEFT": Nach links bewegen
- Der Client lässt die Spielfigur NICHT ins Wasser fallen
- Der Client bewegt sich zur Schriftrolle, wenn er sie sieht
- Der Client bewegt sich zur gegnerischen Burg, nachdem er die Schriftrolle eingesammelt hat
- Ansonsten erkundet der Client die Landschaft - auf der Suche nach Schriftrolle bzw. Burg
- Alle Verbindungen werden sauber geschlossen
- JavaDoc/DocString-Kommentare sowie Dokumentation (Sphinx/JavaDoc) sind vorhanden
- **Protokoll** über den implementierten Algorithmus

Erweiterungen (30%):

- Arbeite dich in den Code des Servers ein
- Erweitere den Server, sodass beliebig große Spielfelder unterstützt werden!
- Sowohl die Größe des Spielfeldes als auch die anderen Parameter (Anzahl an Waldfeldern etc.) sind über den GUI konfigurierbar

Zwischenabgabe (20%):

Am 23.1. wird es eine Zwischenabgabe geben, welche zur Note zählt!

Viel Erfolg!

2 Ergebnisse

```
def doEverythingForMeSimple():
    i = 0
    j = 0
    fortfahren = True
    while(fortfahren):
        time.sleep(0.25)
        #ueberpruefen ob Lake am naechsten Feld
        if (len(fielddata)==18):
            print(fielddata[10])
            if (fielddata[10]=='L' and j!=4):
                    goUp()
                    j += 1
                    i -= 1
            else:
                # Jeden Durchlauf eins nach unten
                if (i >= 9):
                    if (fielddata[14] == 'L'):
                        goRight()
                        #i=-1
                    else:
                        goDown()
                        i = 0
                elif (j >= 4):
                    # wenn ausgewichen nach oben, dann wieder nach
unten gehen
                    goDown()
                    i -= 1
                    j = 0
                else:
                    goRight()
        elif (len(fielddata)==50):
            print(fielddata[26])
            if (fielddata[26] == 'L' and j!=4):
                    goUp()
                    j += 1
                    i -= 1
            else:
                # Jeden Durchlauf eins nach unten
                if (i >= 9):
                    if (fielddata[34] == 'L'):
                        goRight()
                        #i = -1
                    else:
                        goDown()
                        i = 0
                elif (j >= 4):
                    # wenn ausgewichen nach oben, dann wieder nach
unten gehen
                    goDown()
                    i -= 1
                    j = 0
```

```
                else:
                    goRight()
            elif (len(fielddata)==98):
                print(fielddata[50])
                if (fielddata[50] == 'L' and j!=4):
                        goUp()
                        j += 1
                        i -= 1
                else:
                    # Jeden Durchlauf eins nach unten
                    if (i >= 9):
                        if (fielddata[62] == 'L'):
                            goRight()
                            #i = -1
                        else:
                            goDown()
                            i = 0
                    elif (j >= 4):
        # wenn ausgewichen nach oben, dann wieder nach unten gehen
                        goDown()
                        i -= 1
                        j = 0
                    else:
                        goRight()
            else:
                pass
            i += 1
            if (j>0):
                j += 1
            print("i: " + str(i))
            print("j: " + str(j))
            rec_fields(clientsocket)
            if not fielddata:
                fortfahren = False
                clientsocket.close()
```

2.1 Beschreibung des Algorithmus

Das eigentlich Binding der Aktionen des Charakteres sind durch Methoden wie goUP() / DOWN / LEFT / RIGHT ersetzt worden. Diese Methoden werden im Algorithmus aufgerufen. Zuerst wird überprüft ob am nächsten Feld eine Lake ist oder nicht, wenn nicht dann wird eine Bewegung nach rechts ausgeführt sowie bei jedem Durchlauf. Wenn sich im Weg ein Hindernis befindet, dann wird das Element durch ein UP und RIGHT und dann wieder Down umgangen.

Mit der Laufvariable j wird verhindert, dass alle Bewegungsbefehle zugleich

ausgeführt werden und somit Schritte übersprungen werden. Am Ende eines Zuges wird noch eine Statusmeldung ausgeben.