HearthStats

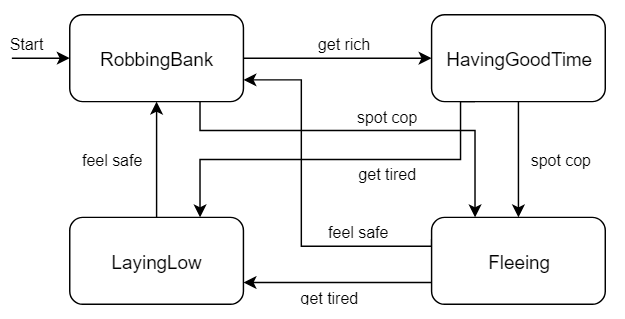
Datastructures and Algorithms

Scott Marchant (500803802) & Iris Oostra (500801711)

Hogeschool van Amsterdam  IG202

# Question 1)

Add two new state transitions to the diagram above. Name these transitions and draw out the new diagram.



# Aanpak

We hebben twee states bedacht, die niet te veel op de anderen leken en bedacht hoe we ze aan de andere states konden koppelen.

# Antwoord

RobbingBank

BeingTipsy

BeingInJail

LayingLow

Fleeing

HavingGoodTime

Get tired

Spot cop

Feel safe

Get rich

Spot cop

Feel safe

Get tired

Start

Get caught

Get caught

Escape

Tell too much

Go to bar

Get bored

Get bored

# Code

Hier was geen code voor nodig.

# Output

Niet van toepassing, omdat er geen code bij nodig was.

# Question 2)

Implement your new state machine using switch statements, a state transition table or the state design pattern. Build it in such a way that the user can change the NPC's state by writing a string ("get rich", "spot cop", etc.) in the Console.

# Aanpak

We bedachten dat het handig is als je switches gebruikt, zodat er dingen gebeuren op basis van de huidige state. Ook bedachten wij dat het handig is om meerdere functies te gebruiken, zodat je niet dezelfde code tachtig keer opnieuw hoeft te typen.

# Antwoord

Op basis van wat we bij Aanpak hebben geschreven hebben wij de code opgebouwd.

# Code

#include <iostream>

#include <string>

using namespace std;

using std::ostringstream;

typedef enum StateIDs{

StartGame,

RobbingBank,

HavingGoodTime,

LayingLow,

Fleeing,

BeingInJail,

BeingTipsy

} State;

typedef enum ActionIDs {

Start,

GetRich,

SpotCop,

GetCaught,

GetTired,

GoToBar,

FeelSafe,

GetBored,

Escape,

TellTooMuch

} Action;

State currentState = StartGame;

Action currentAction;

string actionInput;

string actionPerformingText;

void playGame();

State stateManager(State beforeState, Action performingAction);

string stateMessage();

Action actionManager(string actionInputGiven);

int main()

{

cout << "Press enter to begin." << endl;

if (cin.get() == '\n') {

currentAction = actionManager("");

currentState = stateManager(currentState, currentAction);

cout << actionPerformingText << endl;

cout << stateMessage() << endl << endl << endl;

playGame();

}

}

void playGame() {

cout << "What's next? (type exit to quit the game)" << endl;

getline(cin, actionInput);

currentAction = actionManager(actionInput);

currentState = stateManager(currentState, currentAction);

cout << actionPerformingText << endl;

cout << stateMessage() << endl << endl << endl;

/\*if (cin.get() == 'exit') {

return;

}

else {\*/

//cout << "What's next? (type exit to quit the game)" << endl;

playGame();

//}

}

State stateManager(State beforeState, Action performingAction) {

State outputState;

switch(beforeState) {

case StartGame:

if (performingAction == Start) {

outputState = RobbingBank;

actionPerformingText = "StartGame > Start > RobbingBank";

}

else {

outputState = StartGame;

actionPerformingText = "I'm so confused.";

}

break;

case RobbingBank:

switch (performingAction) {

case GetRich:

outputState = HavingGoodTime;

actionPerformingText = "Robbing bank > Get rich > Having good time";

break;

case SpotCop:

outputState = Fleeing;

actionPerformingText = "Robbing bank > Spot cop > Fleeing";

break;

case GetCaught:

outputState = BeingInJail;

actionPerformingText = "Robbing bank > Get caught > Being in jail";

break;

default:

outputState = RobbingBank;

actionPerformingText = "Let's try again";

}

break;

case HavingGoodTime:

switch (performingAction) {

case GetTired:

outputState = LayingLow;

actionPerformingText = "Having good time > Get tired > Laying low";

break;

case SpotCop:

outputState = Fleeing;

actionPerformingText = "Having good time > Spot cop > Fleeing";

break;

case GoToBar:

outputState = BeingTipsy;

actionPerformingText = "Having good time > Go to bar > Being tipsy";

break;

default:

outputState = HavingGoodTime;

actionPerformingText = "Let's try again";

}

break;

case LayingLow:

switch (performingAction) {

case FeelSafe:

outputState = RobbingBank;

actionPerformingText = "Laying low > Feel safe > Robbing bank";

break;

case GetBored:

outputState = BeingTipsy;

actionPerformingText = "Laying low > Get bored > Being tipsy";

break;

default:

outputState = LayingLow;

actionPerformingText = "Let's try again";

}

break;

case Fleeing:

switch (performingAction) {

case GetTired:

outputState = LayingLow;

actionPerformingText = "Fleeing > Get tired > Laying low";

break;

case FeelSafe:

outputState = RobbingBank;

actionPerformingText = "Fleeing > Feel safe > Robbing bank";

break;

case GetCaught:

outputState = BeingInJail;

actionPerformingText = "Fleeing > Get caught > Being in jail";

break;

default:

outputState = Fleeing;

actionPerformingText = "Let's try again";

}

break;

case BeingInJail:

switch (performingAction) {

case Escape:

outputState = Fleeing;

actionPerformingText = "Being in jail > Escape > Fleeing";

break;

default:

outputState = BeingInJail;

actionPerformingText = "Let's try again";

}

break;

case BeingTipsy:

switch (performingAction) {

case TellTooMuch:

outputState = BeingInJail;

actionPerformingText = "Being tipsy > Tell too much > Being in jail";

break;

case GetBored:

outputState = RobbingBank;

actionPerformingText = "Being tipsy > Get bored > Robbing bank";

break;

default:

outputState = BeingTipsy;

actionPerformingText = "Let's try again";

}

break;

default:

outputState = StartGame;

actionPerformingText = "Let's start over";

}

return outputState;

}

string stateMessage() {

string outputStateMessage;

switch (currentState) {

case RobbingBank:

outputStateMessage = "I am robbing a bank.";

break;

case HavingGoodTime:

outputStateMessage = "I am having a good time.";

break;

case LayingLow:

outputStateMessage = "I am laying low.";

break;

case Fleeing:

outputStateMessage = "I am fleeing.";

break;

case BeingInJail:

outputStateMessage = "I am in Jail now.";

break;

case BeingTipsy:

outputStateMessage = "I am tipsy.";

break;

default:

outputStateMessage = "I am so confused.";

}

return outputStateMessage;

}

Action actionManager(string actionInputGiven) {

Action actionOutput;

if (actionInput == "Get rich") {

actionOutput = GetRich;

}

else if (actionInput == "Spot cop") {

actionOutput = SpotCop;

}

else if (actionInput == "Get caught") {

actionOutput = GetCaught;

}

else if (actionInput == "Get tired") {

actionOutput = GetTired;

}

else if (actionInput == "Go to bar") {

actionOutput = GoToBar;

}

else if (actionInput == "Feel safe") {

actionOutput = FeelSafe;

}

else if (actionInput == "Get bored") {

actionOutput = GetBored;

}

else if (actionInput == "Escape") {

actionOutput = Escape;

}

else if (actionInput == "Tell too much") {

actionOutput = TellTooMuch;

}

else {

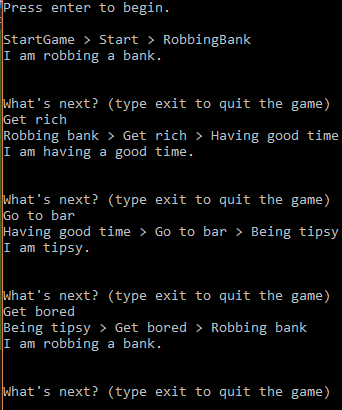
actionOutput = Start;

}

return actionOutput;

}

# Output



# Question 3)

Add output to your program by showing a line of text that indicates the current state or action. For example, while he is robbing banks he may say:

"I'm robbing banks and getting loads of money! Pew pew!"

For the other states and actions you can use the following lines:

"I'm rich enough to have a good time" (for the 'get rich' action)

"I'm having a good time spending my money" (for the HavingGoodTime state) "I see a cop, so I have to start running"

"I'm getting very tired, so I better lay low for a while"

etc.

Or make up your own lines.

# Aanpak

**W**e bedachten tekstregels die ons leuk leken en plaatsten deze op de juiste plekken in de code.

# Antwoord

|  |  |  |  |
| --- | --- | --- | --- |
| From state | Action | To state | Text line |
| RobbingBank | Get rich | HavingGoodTime | "I'm rich enough to have a good time" |
| RobbingBank | Spot cop | Fleeing | "I see a cop, so I have to start running" |
| RobbingBank | Get caught | BeingInJail | “Oh no, the cops are bringing me to my new home!” |
| HavingGoodTime | Get tired | LayingLow | "I'm getting very tired, so I better lay low for a while" |
| HavingGoodTime | Spot cop | Fleeing | "I see a cop, so I have to start running" |
| HavingGoodTime | Go to bar | BeingTipsy | “All drinks on me!” |
| Fleeing | Feel safe | RobbingBank | “Let’s get back to what I was doing” |
| Fleeing | Get tired | LayingLow | "I'm getting very tired, so I better lay low for a while" |
| Fleeing | Get caught | BeingInJail | “Oh no, the cops are bringing me to my new home!” |
| LayingLow | Feel safe | RobbingBank | “Let’s get back to what I was doing” |
| LayingLow | Get bored | BeingTipsy | “A bar is a good safe spot, right? Right??” |
| BeingInJail | Escape | Fleeing | “RUUUNNN!! RUUUUUUUUNNNN!!!” |
| BeingTipsy | Get bored | RobbingBank | “You know what would be funny..? If we go ‘borrow’ something from the bank…hehe” |
| BeingTipsy | Tell too much | BeingInJail | “No! Wait! Those were just stories!” |

|  |  |
| --- | --- |
| Current state | Text line |
| RobbingBank | "I'm robbing banks and getting loads of money! Pew pew!" |
| HavingGoodTime | "I'm having a good time spending my money" |
| Fleeing | “Who said being a thief isn’t a sport? I’m running all day!” |
| LayingLow | “I’ll just wait a minute..” |
| BeingInJail | “So when can I go home? Jail is stupid.” |
| BeingTipsy | “Glug..glug…. hmmmm……. beeeer" |

# Code

State stateManager(State beforeState, Action performingAction) {

State outputState;

switch(beforeState) {

case StartGame:

if (performingAction == Start) {

outputState = RobbingBank;

actionPerformingText = "'Let's start this'";

}

else {

outputState = StartGame;

actionPerformingText = "'I'm so confused'";

}

break;

case RobbingBank:

switch (performingAction) {

case GetRich:

outputState = HavingGoodTime;

actionPerformingText = "'I'm rich enough to have a good time'";

break;

case SpotCop:

outputState = Fleeing;

actionPerformingText = "'I see a cop, so I have to start running'";

break;

case GetCaught:

outputState = BeingInJail;

actionPerformingText = "'Oh no, the cops are bringing me to my new home!'";

break;

default:

outputState = RobbingBank;

actionPerformingText = "'Let's try again'";

}

break;

case HavingGoodTime:

switch (performingAction) {

case GetTired:

outputState = LayingLow;

actionPerformingText = "'I'm getting very tired, so I better lay low for a while'";

break;

case SpotCop:

outputState = Fleeing;

actionPerformingText = "'I see a cop, so I have to start running'";

break;

case GoToBar:

outputState = BeingTipsy;

actionPerformingText = "'All drinks on me!'";

break;

default:

outputState = HavingGoodTime;

actionPerformingText = "'Let's try again'";

}

break;

case LayingLow:

switch (performingAction) {

case FeelSafe:

outputState = RobbingBank;

actionPerformingText = "'Let’s get back to what I was doing'";

break;

case GetBored:

outputState = BeingTipsy;

actionPerformingText = "'A bar is a good safe spot, right? Right??'";

break;

default:

outputState = LayingLow;

actionPerformingText = "'Let's try again'";

}

break;

case Fleeing:

switch (performingAction) {

case GetTired:

outputState = LayingLow;

actionPerformingText = "'I'm getting very tired, so I better lay low for a while'";

break;

case FeelSafe:

outputState = RobbingBank;

actionPerformingText = "'Let’s get back to what I was doing'";

break;

case GetCaught:

outputState = BeingInJail;

actionPerformingText = "'Oh no, the cops are bringing me to my new home!'";

break;

default:

outputState = Fleeing;

actionPerformingText = "'Let's try again'";

}

break;

case BeingInJail:

switch (performingAction) {

case Escape:

outputState = Fleeing;

actionPerformingText = "'RUUUNNN!! RUUUUUUUUNNNN!!!'";

break;

default:

outputState = BeingInJail;

actionPerformingText = "'Let's try again'";

}

break;

case BeingTipsy:

switch (performingAction) {

case TellTooMuch:

outputState = BeingInJail;

actionPerformingText = "'No! Wait! Those were just stories!'";

break;

case GetBored:

outputState = RobbingBank;

actionPerformingText = "'You know what would be funny..? If we go ‘borrow’ something from the bank…hehe'";

break;

default:

outputState = BeingTipsy;

actionPerformingText = "'Let's try again'";

}

break;

default:

outputState = StartGame;

actionPerformingText = "'Let's start over'";

}

return outputState;

}

string stateMessage() {

string outputStateMessage;

switch (currentState) {

case RobbingBank:

outputStateMessage = "'I'm robbing banks and getting loads of money! Pew pew!'";

break;

case HavingGoodTime:

outputStateMessage = "'I'm having a good time spending my money'";

break;

case LayingLow:

outputStateMessage = "'I’ll just wait a minute..'";

break;

case Fleeing:

outputStateMessage = "'Who said being a thief isn’t a sport? I’m running all day!'";

break;

case BeingInJail:

outputStateMessage = "'So when can I go home? Jail is stupid.'";

break;

case BeingTipsy:

outputStateMessage = "'Glug..glug…. hmmmm……. beeeer'";

break;

default:

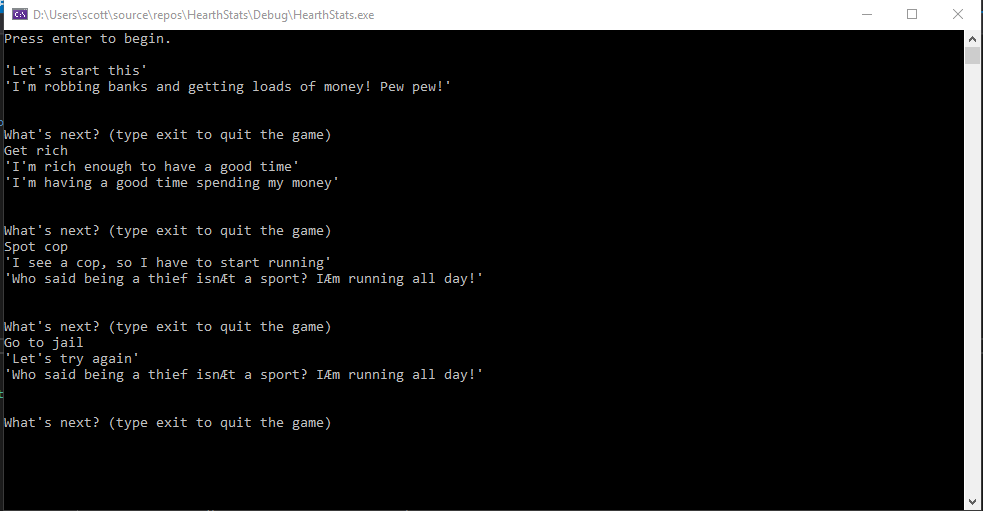
outputStateMessage = "'I am so confused'";

}

return outputStateMessage;

}

# Output



# Question 4)

Add variables to your NPC for wealth, distanceToCop and strength. Wealth increases while the NPC is robbing banks, but decreases while he is having a good time or fleeing. distanceToCop can suddenly change to 0 during robbing a bank or having a good time, which causes the NPC to start fleeing. strength decreases during robbery, having a good time and fleeing, but increases during laying low. Continue this way and make sure that each state transition depends on one or more of these variables, and draw out your new diagram.

# Aanpak

**W**e hebben per lijn bedacht wat er ongeveer qua variabele-waarden bij passen en ervoor gezorgd dat er per state niet twee lijnen bij dezelfde waardes horen. Ook hebben we per blokje verder bedacht wat wij logisch vonden dat deze state met de variabelen zou doen.

# Antwoord

**RobbingBank**

wealth++

distanceCop RND(0)

strength--

**BeingTipsy**

wealth--

distanceCop RND(0)

strength--

**BeingInJail**

wealth 0

distanceCop--

strength++

**LayingLow**

wealth

distanceCop++

strength++

**HavingGoodTime**

wealth--

distanceCop RND(0)

strength--

**Fleeing**

wealth--

distanceCop RND(0)

strength--

Low strength, wealth med

distanceCop med

distanceCop med

Wealth high

distanceCop 0

distanceCop high, strength high

distanceCop 0, strength low

Start

Strength med, wealth med

distanceCop 0

Strength high

Low strength, distanceCop low

Strength low

Strength med, wealth low

distanceCop high

# Code

int wealth;

int distanceCop;

int strength;

# Output

**D**eze code geeft nog geen output.

# Question 5)

Implement the variables and conditions state transitions that you chose in your NPC.

Add a loop to the program, such state it updates the variables and looks at the current state at each iteration; it then determines whether a state transition should occur.

Also implement a one second pause at each iteration to your program. You can now remove the user input function, as the new states is set only by the variable's values which are updated at each iteration. Adjust your program so that this works and show the output of an example run.

This loop could go on indefinitely. We are going to add a Cop NPC to the HearthStats game that can catch the first NPC, such that the program can end.

# Aanpak

**W**e hebben eerst bedacht hoe we ervoor konden zorgen dat de output van een state steeds anders is, zodat er een random transitie naar een volgende state kon gebeuren. Daarna bedachten we dat het slim was om het hele action-gedeelte uit de code te halen en gewoon het programma te laten kijken wat de variabelen momenteel zijn en wat er op basis daarvan moet gebeuren.

# Antwoord

**W**e hebben als eerste een for-loop gemaakt die in 60x ervoor zorgt dat de variabele willekeurig genoeg verandert worden op basis van de huidige state. Daarna hebben we ook ervoor gezorgd dat er 1 seconde pauze is voordat alles op het scherm wordt weergegeven. Hierna hebben we in plaats van de methode die checkte wat de huidige state was en wat de huidige action was, hebben we gecheckt op de huidige state en de huidige variabele-waardes.

# Code

#include <iostream>

#include <string>

#include <stdlib.h>

#include <stdlib.h>

#include <time.h>

#include <chrono>

#include <thread>

using namespace std;

using namespace std::this\_thread;

using namespace std::chrono\_literals;

using std::chrono::system\_clock;

typedef enum StateIDs{

StartGame,

RobbingBank,

HavingGoodTime,

LayingLow,

Fleeing,

BeingInJail,

BeingTipsy

} State;

//Max value = 100 for wealth and strength, 10 for distanceCop

int wealth;

int distanceCop;

int strength;

int fps = 60;

State currentState = StartGame;

string actionInput;

string actionPerformingText;

void playGame();

void checkMax();

void varChangeByState();

State stateManager(State beforeState, int currentWealth, int currentStrength, int currentDistanceCop);

string stateMessage();

int main() {

srand(time(NULL));

cout << "Press enter to begin." << endl;

if (cin.get() == '\n') {

varChangeByState();

checkMax(); //Check whether the variables haven't exceeded their maximum and minimum values

currentState = stateManager(currentState, wealth, strength, distanceCop);

cout << actionPerformingText << endl;

cout << stateMessage() << endl << endl << endl;

playGame();

}

}

void playGame() {

for (int i = 1; i < fps; i++) {

varChangeByState();

checkMax(); //Check whether the variables haven't exceeded their maximum and minimum values

}

sleep\_until(system\_clock::now() + 1s);

currentState = stateManager(currentState, wealth, strength, distanceCop);

cout << "Wealth: " << wealth << endl << "Strength: " << strength << endl << "Distance to cop: " << distanceCop << endl << endl;

cout << actionPerformingText << endl << endl;

cout << stateMessage() << endl << endl << endl << endl;

/\*if (cin.get() == 'exit') {

return;

}

else {\*/

//cout << "What's next? (type exit to quit the game)" << endl;

playGame();

//}

}

void checkMax() {

if(strength > 100) {

strength = 100;

} else if(strength < 0) {

strength = 0;

}

if(distanceCop > 10) {

distanceCop = 10;

} else if(distanceCop < 0) {

distanceCop = 0;

}

if(wealth > 100) {

wealth = 100;

} else if(wealth < 0) {

wealth = 0;

}

}

void varChangeByState(){

switch(currentState) {

case StartGame:

wealth = 0;

strength = 100;

distanceCop = rand() % 9;

break;

case RobbingBank:

wealth += rand() % 10 + 1;

strength--;

distanceCop = rand() % 9;

break;

case HavingGoodTime:

wealth -= rand() % 8 + 1;

strength -= rand() % 8 + 1;

distanceCop = rand() % 9;

break;

case LayingLow:

wealth = wealth;

strength += rand() % 10 + 1;

distanceCop = rand() % 4 + 7;

break;

case Fleeing:

wealth--;

strength -= rand() % 8 + 1;

distanceCop = rand() % 9;

break;

case BeingInJail:

wealth = 0;

strength += rand() % 2 + 1;

distanceCop = 0;

break;

case BeingTipsy:

wealth -= rand() % 2 + 1;

strength -= rand() % 2 + 1;

distanceCop = rand() % 4;

break;

default:

wealth = 0;

strength = 100;

distanceCop = rand() % 9;

}

}

State stateManager(State beforeState, int currentWealth, int currentStrength, int currentDistanceCop) {

State outputState;

switch (beforeState) {

case StartGame:

if (currentWealth == 0 && currentStrength == 100) {

outputState = RobbingBank;

actionPerformingText = "\"Let\'s start this\"";

}

else {

outputState = beforeState;

actionPerformingText = "\"I\'m so confused\" + ERROR CHECK STARTGAME";

}

break;

case RobbingBank:

if (currentWealth >= 60 && currentDistanceCop > 5) {

//GetRich

outputState = HavingGoodTime;

actionPerformingText = "\"I\'m rich enough to have a good time\"";

}

else if (currentDistanceCop > 0 && currentDistanceCop <= 5) {

//SpotCop

outputState = Fleeing;

actionPerformingText = "\"I see a cop, so I have to start running\"";

}

else if (currentDistanceCop == 0) {

//GetCaught

outputState = BeingInJail;

actionPerformingText = "\"Oh no, the cops are bringing me to my new home!\"";

}

else {

outputState = beforeState;

actionPerformingText = "\"Let\'s try again\" + ERROR CHECK ROBBINGBANK";

}

break;

case HavingGoodTime:

if (currentWealth >= 40 && currentWealth <= 70 && currentStrength < 40) {

//GetTired

outputState = LayingLow;

actionPerformingText = "\"I\'m getting very tired, so I better lay low for a while\"";

}

else if (currentDistanceCop == 0) {

//SpotCop

outputState = Fleeing;

actionPerformingText = "\"I see a cop, so I have to start running\"";

}

else if (currentStrength < 40) {

//GoToBar

outputState = BeingTipsy;

actionPerformingText = "\"All drinks on me!\"";

}

else {

outputState = beforeState;

actionPerformingText = "\"Let\'s try again\" + ERRO CHECK HAVINGGOODTIME";

}

break;

case LayingLow:

if (currentStrength >= 65 && currentDistanceCop >= 7) {

//FeelSafe

outputState = RobbingBank;

actionPerformingText = "\"Let\'s get back to what I was doing\"";

}

else if (currentStrength < 65 && currentWealth >= 40 && currentWealth <= 70) {

//GetBored

outputState = BeingTipsy;

actionPerformingText = "\"A bar is a good safe spot, right? Right??\"";

}

else {

outputState = beforeState;

actionPerformingText = "\"Let\'s try again\" + ERROR CHECK LAYINGLOW";

}

break;

case Fleeing:

if (currentDistanceCop < 7 && currentDistanceCop > 0) {

//GetTired

outputState = LayingLow;

actionPerformingText = "\"I\'m getting very tired, so I better lay low for a while\"";

}

else if (currentDistanceCop >= 7) {

//FeelSafe

outputState = RobbingBank;

actionPerformingText = "\"Let\'s get back to what I was doing\"";

}

else if (currentDistanceCop == 0 && currentStrength < 50) {

//GetCaught

outputState = BeingInJail;

actionPerformingText = "\"Oh no, the cops are bringing me to my new home!\"";

}

else {

outputState = beforeState;

actionPerformingText = "\"Let\'s try again\" + ERROR CHECK FLEEING";

}

break;

case BeingInJail:

if (currentStrength >= 70) {

//Escape

outputState = Fleeing;

actionPerformingText = "\"RUUUNNN!! RUUUUUUUUNNNN!!!\"";

}

else {

outputState = beforeState;

actionPerformingText = "\"Let\'s try again\" + ERROR CHECK BEINGINJAIL";

}

break;

case BeingTipsy:

if (currentStrength < 40) {

//Tell too much

outputState = BeingInJail;

actionPerformingText = "\"No! Wait! Those were just stories!\"";

}

else if (currentStrength >= 40 && currentWealth < 40) {

//Get bored

outputState = RobbingBank;

actionPerformingText = "\"You know what would be funny..? If we go \'borrow\' something from the bank…hehe\"";

}

else {

outputState = beforeState;

actionPerformingText = "\"Let\'s try again\"";

}

break;

default:

outputState = StartGame;

actionPerformingText = "\"Let\'s start over\" + ERROR CHECK DEFAULTSTATE";

}

return outputState;

}

string stateMessage() {

string outputStateMessage;

switch (currentState) {

case RobbingBank:

outputStateMessage = "\"'I\'m robbing banks and getting loads of money! Pew pew!\"";

break;

case HavingGoodTime:

outputStateMessage = "\"I\'m having a good time spending my money\"";

break;

case LayingLow:

outputStateMessage = "\"I\'ll just wait a minute..\"";

break;

case Fleeing:

outputStateMessage = "\"Who said being a thief isn\'t a sport? I\'m running all day!\"";

break;

case BeingInJail:

outputStateMessage = "\"So when can I go home? Jail is stupid.\"";

break;

case BeingTipsy:

outputStateMessage = "\"Glug..glug.... hmmmm..... beeeer\"";

break;

default:

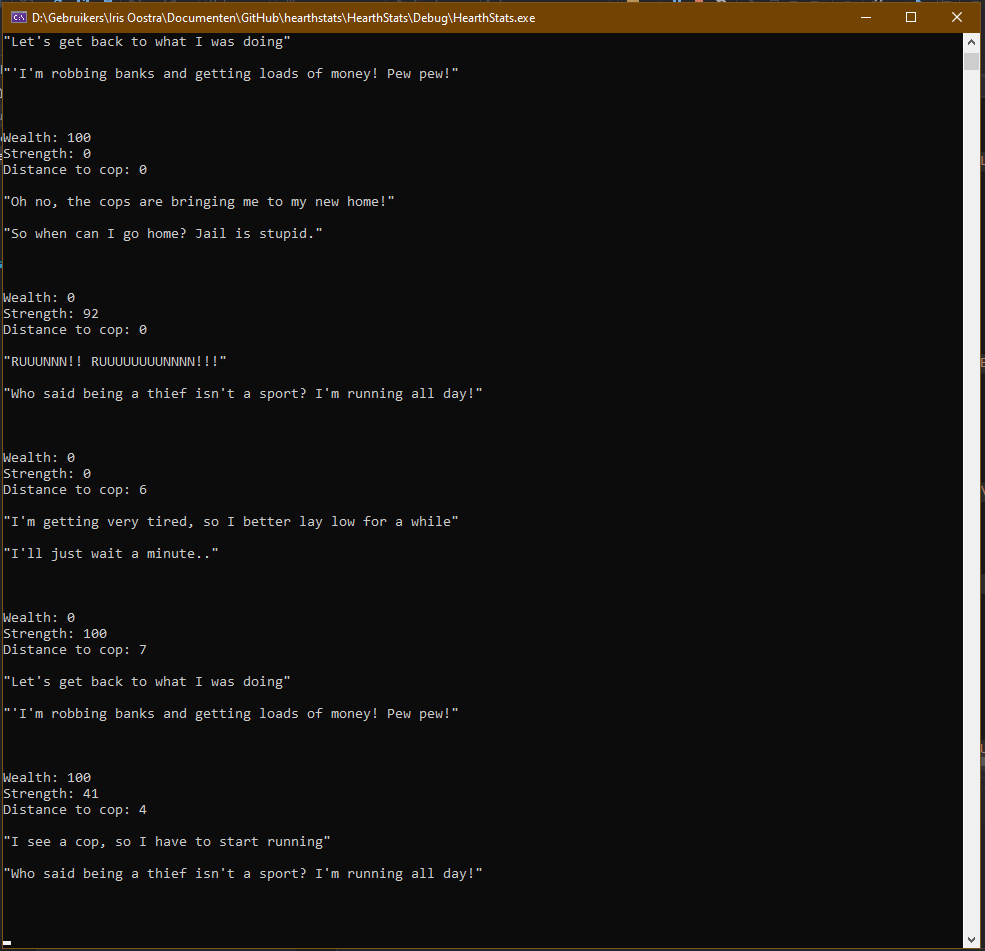
outputStateMessage = "\"I am so confused\" + ERROR CHECK DEFAULTMESSAGE";

}

return outputStateMessage;

}

# Output



# Question 6)

Design a state diagram for a second NPC (Cop) with three possible states: OffDuty, OnStakeOut and Chasing.

Design logical state transitions that depend on the variable dutyTime: the value of this variable increases during OnStakeOut and Chasing until it reaches a certain value, at which point the state transitions to OffDuty, when it starts decreasing again until 0.

The Cop initially starts in the OnStakeOut state. When the former NPC starts robbing banks, the Cop will at some point start Chasing. The first NPC then starts fleeing, which changes the distanceToCop value.

Draw out the full Cop state diagram and design logical transitions between states such that he is able to catch the first NPC and the program terminates.

# Aanpak

**W**e hebben als eerste alle onderdelen die in de opdracht beschreven waren toegevoegd en daarna de ontbrekende dingen zo ingevuld dat het logisch is binnen ons programma, in ons programma komt de robber namelijk ook in de gevangenis terecht en kan het soms zijn dat hij nog ontsnapt. Dit hebben wij opgelost door twee verschillende “gevangen” te hebben. Als de robber gevangen is en de dutyTime op is, dan terminate het programma en heeft de cop “gewonnen”. Als de dutyTime nog niet op is en de robber zit nog in de gevangenis, dan kan het nog zijn dat hij ontsnapt en het programma nog even verder runt.

# Antwoord

**OffDuty**

dutyTime--

**OnStakeOut**

dutyTime++

**Chasing**

dutyTime++

Start

Robber = RobbingBanks

dutyTime 0

dutyTime max AND robber caught

# Code

int dutyTime;

typedef enum CopStateIDs{

OnStakeOut,

Chasing,

OffDuty

} CopState;

# Output

**D**eze code geeft nog geen output.

# Question 7)

Implement the Cop NPC in your existing program loop. Make sure the Cop also outputs a line indicating its current state or actions, for example:

"Duty time's over, I'm headed home"

"Relaxing on the sofa, watching Crime Scene Investigation" "Hold it right there buddy" etc.

# Aanpak

# Antwoord

# Code

# Output

# Question 8)

Show the output of an example run that terminates by itself, and showing alternating lines said by both NPC's.

# Aanpak

# Antwoord

# Code

# Output