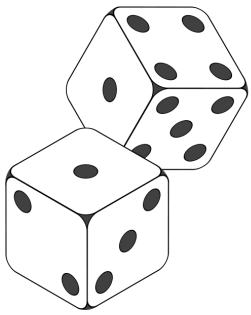
Taking Back The Control Over The Dice

First Steps to Digitalizing Classic Board Games



M.Heijn 2663098 | J.Zoryk 2663347 | M.Chong 2666136

Supervisors: Dr.Natalia Silvis-Cividjian and Koen Kahlman

Pervasive Computing Project

VU VRIJE UNIVERSITEIT AMSTERDAM

Faculty of Sciences
Vrije Universiteit
Amsterdam, Netherlands

23 October 2022

Contents

1 Introduction

Board games are not only regarded as a form of entertainment, where people can interact and connect with each in a social manner, they are also a means to promote abstract thinking and logical processes, which is extremely helpful in developing one's physical and mental health. Psychologists from the University of Edinburgh tested the cognitive ability of more than 1000 people at the age of 70, and repeated these tests every three years until the age of 79. They found that people who regularly played non-digital games scored better on memory and thinking tests (Altschul and Deary, 2020).

Furthermore, in some cases, classical board games may not be accessible for all. Namely that some board games in the market contain a wide range of pieces in order to play the game. One example is the game 'Monopoly', in which each player needs 'money', 'properties' and 'services' in order to play the game as intended. Note that sufficient space is also required to play the game, which may not be an option for some people. Suppose now that someone with a disability such that it affects their motor control or limits their arm reach wants to play a similar type of board game because they see their friends playing it and want to join the fun. It is not long before one can see that for such an individual to play classic board games, it would be difficult without some assistance from his/her friends or someone else. This, in turn makes for a negative overall experience of playing board games and potentially resulting in a somewhat traumatic experience causing them not to play board games at all. Meaning that they lose a chance to develop their cognitive mental abilities in a entertaining and social way.

We believe that this is not the right approach and would like to find a way to address such problems. Allowing anyone to 'take back the control over the dice' again regardless of their disabilities. Therefore this goal of this project is to design a system that reduces the logistical complexity of current board games, while maintaining key physical attributes for the given game. Due to the strict and short time period we have to conduct this project, we task ourselves to make a voice controlled computer version of 'Snakes and Ladders', with a physical dice reader.

2 Literature Study

In this section of the document we investigate the validity of our project idea, namely we want to see if it has been done before and find papers that can assists us while conducting this project.

One of our first observation in the literature study is that we see that there are digital version of most of the classic board games, this includes our games "Snakes and Ladders" virtual Snakes and Ladders.. Here we see that the online version models every aspect of the game, from the board and players' positions to that of each dice roll. However, when playing this online version we find that it lacks entertainment for the players, as now all the player has to do to play the game is click a button. This does not involve any skill or luck of the player resulting in a boring game. Our analysis is supported by a paper that highlights the importance of dice throwing in order for the game to be enjoyable in specifically the table top game. With the paper stating we argue that the physicality of dice has a positive effect on player's experience and enjoyment of the game (Carter et al., 2014).

Many players share the belief that proper grip on the dice, or speaking to them, or perhaps kissing them will improve their luck. In our investigation we found a papers that indicates that the rolling of a dice is not completely random (Nagler and Richter, 2008 and Stein, 2012). Moreover, that it is actually a skill of the player in rolling of the dice, that allows them to win. Hence any virtual dice thrower, will admit such physical properties of the game, so in order for the player to maintain the illusion of control, of participation rather than observation we require

the game to involve physical dice throwing. Therefore we see the importance of physical dice throwing in our board game project.

During our research we found that reintroducing the physical dice throwing in a computer game is unique concept and might be the beginning of a new set of computer games.

Another aspect that we came across is the need of board game aimed towards people with disabilities. As we found a common question that appears on most forums that are dedicated to board game is that someone is asking for recommendation for games to be played with their friend that has a disability. For an example here is a question asked :

> *"My friend has muscular dystrophy and can't move independently. He doesn't have any cognitive problems. I'm looking for a board game we could play together. Something that doesn't involve writing stuff down or manipulating game pieces. A game where it's easy to describe to someone else what he wants to do on his turn would be ideal."* (boardgamegeek, 2022).

With our project we hope to give more options, ultimate the goal is so that they can play any board game they wanted with our system that we develop during this project.

Over the pervasive computing course that we took and also reading the associated literature, we become exposed to the current field of technologies and method in order for us to complete this project (Silvis-Cividjian, 2017).

3 Concept of Operations

The concept of this project is to "take back the control over games dice". As we stated above, is the physical handling of the dice essential for board games. Introducing this in a computer game has two major benefits. Firstly, make board games available for the computer generation and secondly make board games accessible to less enabled. Potential stakeholders are then the game industry itself, the classical computer gamer, but also the less enabled player and care workers. Since most board games are social games parents and schools may also be interested.

With this project we want to

+ develop a game of snakes and ladders for which only voice commands and physical dice throw is necessary,
+ capture dice throw and present values of the dice,
+ develop voice command recognition like "'yes" and "no" to interact with the system and
+ to play a game of snakes and ladders.

A possible scenario of such a game is shown in Figure 1 on the following page and is described as follows.

Start System is idle state.

1. The system detects that there is a player by a movement detection in front of a camera.
2. The system asks who wants to play. The names of the players are recorded. The system asks if the recorded name is correct and if there are more players. The system has to respond to spoken command like "yes" and "'no".
3. Who starts the game. The system invites the named player to throw the dice. The value of the throw is recorded and the player with the highest value can begin the game. In a tie the concerned players have to throw again.

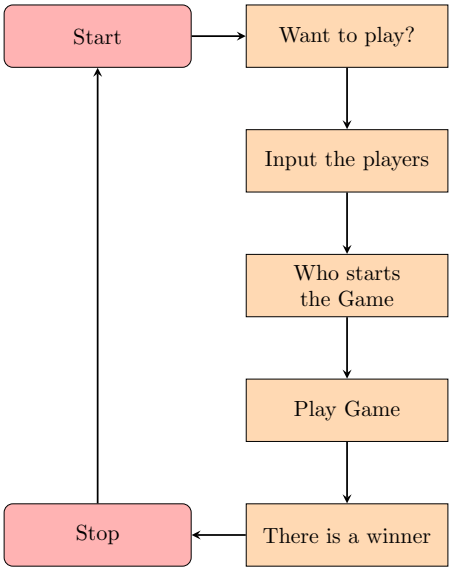


Figure 1: Main concept of the system

4 A play of Snakes and Ladders is played. The system asks the named player to throw the dice, the value of the throw is given and the player's pawn is moved accordingly.

5 At the end of the game there is a winner. The system goes back to 'start' in the idle state.

4 System Requirements

Here we adopt the MoSCoW approach to listing all the requirements of the system, from which we are able to priories our time by working on the key parts of the systems in a systematic manner, see Table 1.

The structure of the game is shown in Figure 2. As inputs there are voice controlled commands like "Yes" and "No" and image related inputs like the throw of a dice. Outputs are spoken commands by the computer and the image of the game board with the pawns of the players. Hence, the system can be in four states. It wants an input either a spoken command (1) or a

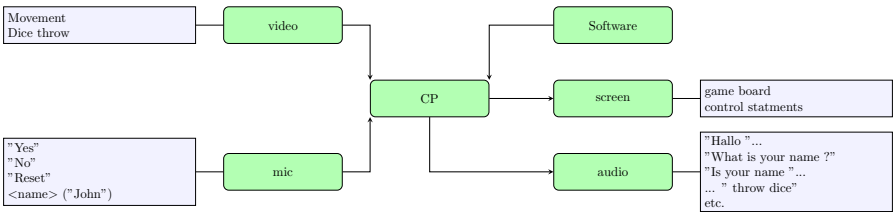


Figure 2: Block diagram of the game

video image (2) or gives an output either a picture of the game board (3) or a spoken text (4).

The system must in state 1 one recognize a spoken word. The recorded spoken word must be filtered from the ambient sounds. To recognize the word the obtained recorded data must be

Requirements	Must	Should	Could	Won't
Functional	• Read the dots on a dice. • Listen to voice commands • Be a digital version of the game snakes and ladders	• Reads dice in different lighting environments • Listens to voice commands in presence of background noises	• Improved voice detection for different people. • Dice rolling in larger area, rather than cup. • Game sounds for game events	• Reading of two or more dices • Large list of command words • Advance graphics for game
Non-functional	• Dice reading run time within 5s • Voice command accuracy of 80 • Game to be processed in real time	• Detect movement of the dice, then run dice reader when it is stopped	• Read out players names and information on who's turn it is	• Optimized software • Game music

Table 1: MoSCoW table for project system requirements.

classified, using a trained neural network in order to determine whether 'yes', 'no' or 'reset' is said.

In state 2 a digital camera takes pictures in time. Depending on if a movement or a picture of the dice is needed it should return an image when there is huge or small difference between two consecutive pictures. When a picture of a dice is given (a small difference) it should be analyzed and the number of the throw should be extracted.

In state 3 the system should preset on screen an appealing image of the game board where the state of the game, placement of the pawns and other items, is clearly visible.

Finally, the system should play prerecorded texts to coach the players in state 4.

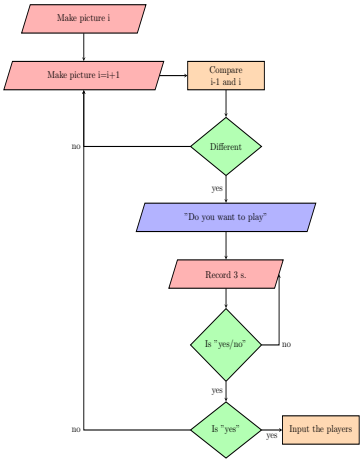
5 System Design

In addition to understanding what our system should do, this section discusses the hypothetical inner workings of the program and the components used to build such a system. Later in the "Testing" section, we shall expand on how the code was actually built.

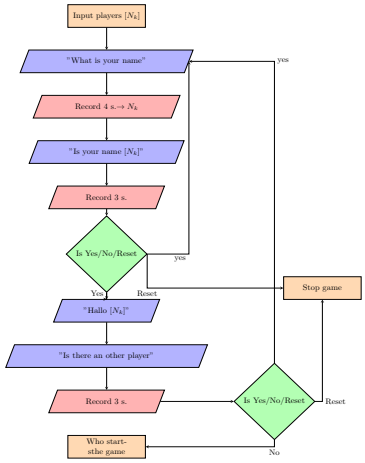
5.1 Components

The following list of basic components or its equivalent are necessary to build our system:

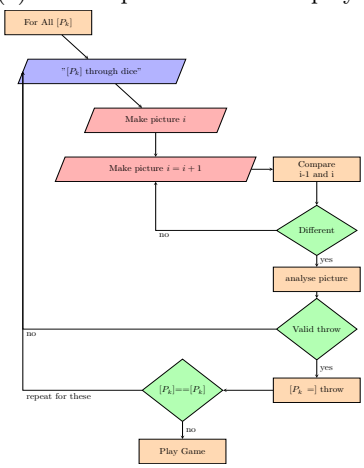
- An **LED display** (laptop, PC, tablet etc.) in order to display the relevant information to the players, such as the score and position.
- A **digital camera** (most web cameras) to capture real time images of the dice in a suitable resolution and compute the value.
- A **digital microphone** with resolution sufficient enough to distinguish between vowel sounds of the voice commands such as, "Yes", "No" and "Reset"; and to capture audio voice in real time.



(a) Subcomponent "Want to play"



(b) Subcomponent "Input players"



(c) Subcomponent "Who starts the game"

- A **programmable processor** (hardware device) that is capable of computing the required calculations of the software in real time and to allow the system to provide real time updates to the players.

5.2 Algorithms, Features and More

Since the project was modular in nature, we separated the system into three main modules: sound (audio/voice) processing, video (image) processing and the game itself. These modules were then later combined to create the whole.

5.2.1 Audio (Voice) Processing

As explained in the previous sections, the system has to be listening at the correct time and accurately respond to the correct user commands. Thankfully, due to the nature of the game progression from computer command to user command to displaying the relevant results on the screen, there is a short delay between each player's turn; which, therefore, does not require the

system to be constantly listening and helps to reduce overall load on the processor.

The first step is to create a function `getsound_vector`, which records the user command for a certain time and extracts the relevant sound with as little background noise as possible while retaining quality. As such, it should reuse the `audiorecorder`, `recordblocking`, and `getaudiodata` functions from Lab 1, and take the moving average of the signal to reduce background noise. Additionally, we should use the Fast Fourier Transform (`FFT`) from Lab 3 to find the relevant amplitudes and to extract the most important peaks (`getpeaks`) in the frequency spectrum.

Finally, since we are interested in identifying a user's commands among other sounds, this naturally translates to a classification problem. The problem being: based on the frequency spectrum of the acquired vowels, which vowels are they? For this purpose, a rule-based classifier and a simple feed-forward neural network was implemented. The highest peaks obtained from the user commands shall be used as features for the neural network (`learn` and `traindata`).

5.2.2 Image Processing

For this module, we want the system to recognize a changed environment and record the values of a new dice roll from the next player.

For the system to recognize the changed environment, we should create a function `movement` (17) that compares one state to another. This can be achieved by first taking an initial image (`acquire_image`) using the digital camera, converting it to greyscale and to a small resolution, taking a subsequent image with the same specifications then finally comparing the absolute values of the pixels from each other. If the absolute values of these images are bigger than some threshold value, then it is considered different or having movement. Otherwise, if the absolute values are below some threshold, then there is no movement. When the intended movement or rest is reached, an image is taken by the `acquire_image` function. Basically, we are observing the changes in light in a focused area such that changes in brightness corresponds to movement.

Once the system detects movement and focuses on the new image, which contains a new dice roll by the next player, a function `dice_count` (p.18) should segment the correct values on the dice using some threshold and store the dice value. To achieve these tasks, we should use a black and white image (converted with `im2bw`), calculate the threshold of the image using the Otsu method or by experimenting, use the `strel` function to create a structuring element, utilize the morphological operation opening (`imopen`) with some radius to remove imperfections from the image, create the complement of the image (`imcomplement`), again applying the opening operation with the size of the dice, then finally using the connected components labeling algorithm (`bwlabel`) to label the dice values (blobs). If the labelled dice values are between 1 to 6, then store the result, otherwise repeat the same process again.

5.2.3 The Game Itself

Lastly for the game module, since we do not want a user to be bored playing the game, it should be interactive and visually appealing. One way to achieve this is to use a template of an already existing snakes and ladders game (n.d.) and correctly map the squares to a matrix of equivalent size in Matlab (`play_game2`). From this, the game should take inputs from the sound and image processing modules, start each player at the 0^{th} position and move their markers by adding their current position with the respective dice values until they reach the end (position ≥ 100). If statements are used to account for players moving up the ladders or going down the snakes.

Once we have the board, we include some sounds to make it interactive. A sound is provided for each action in the game: a ladder (`up.wav`), a snake (`down.wav`) and a step (`step.wav`).

6 Project Plan

The project was separated into three parts, the first part of the project was to develop and build the system, this involves writing the software and integrating hardware. The second part was to generate the required documentation, such as Gantt charts andsystem diagrams, which assists in the development and delivery of the system. The final part was to generate and present a presentation to our supervisors. In order to manage the various parts, we used a Gantt chart Figure 4 to organise our deadlines and work division over the course of the project.

The development of the system was broken into the three core modules that are discussed as in System Design section [5.2]. Naturally we allocated a module to each member of our group to work on and develop. During meetings each persons progress was discussed, this allowed us to assist each other with any problem we might have in development and keeping us on schedule. We utilised Matlab Online to create a shared project folder in which we can make and edit code on the cloud, so everyone had access to the most recent code versions. Similar, we used Overleaf for an online cloud editor that allowed us to collaborate on documentation generating together while keeping track of document versions. The method in which we organised the project allowed us to work on different parts independently, meaning that we could work to each other strengths.

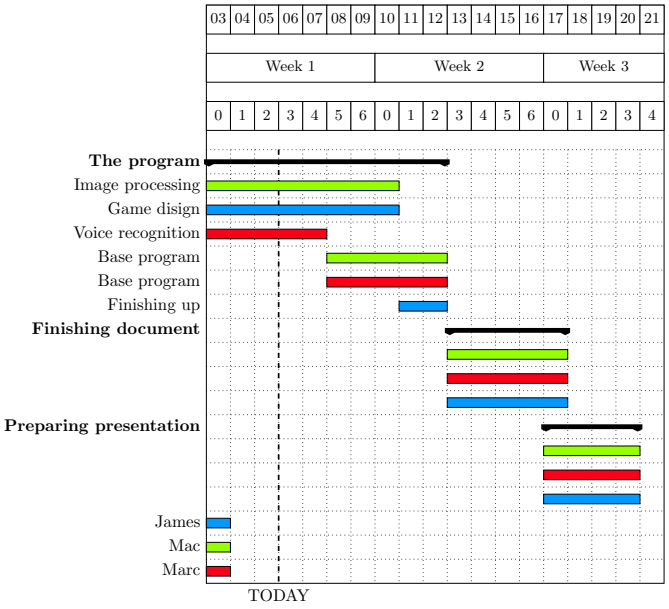


Figure 4: Gantt plot of the project planning.

7 Testing

Because the project was modular of structure, we tested first the modules separately. Finally we tested the whole system.

7.1 Voice Input

The first thing we did was writing a module to record a word (`getsound_vector`, p.20). This function records for a certain time and takes only the first peace sound above a certain level (`ms`), such as detemined by

```
s1=movmean(abs(sound),ceil(Fs/50)); % take ABS signal and simple filter above 50
    Hz
bg=mean(s1(1:Fs/4)); % take first 0.25 sec. as background
ms=bg*2; % take everything above 2x background.
```

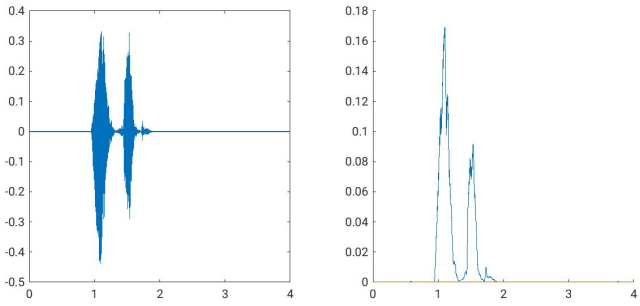


Figure 5: The recording of the word "Reset"
Left panel *are the raw wave data. The* ***right panel*** *are the absolute filtered data.*

In Figure 5 the word *reset* is recorded. The problem for this word is that it only gave the first part (*re*). So we allowed to have 0.25 sec below the threshold `ms`.

The recorded word was then analyzed in a similar way, but now allowing to form two pieces. Like Figure 6 on the following page shows. From these pieces we took a vector 500 long is such a way that the spoken vowel was caught. A `FFT` was performed and the 10 most important (highest) peaks were taken.

To feed the neural net work we recorded at least 40 words, randomly chosen from "yes", "no" and "reset", and fed the 10 most important peaks. (`traindatain` p.21) Figure 7 on page 11 shows that the sounds of 'yes' and 'no' are well distinguished. However, the sounds of 'yes' and 'rea' show some overlap. Here some improvement is still necessary.

This module is complete with the `'make_spoken'` (p.22) function, which record commands and saves them as `.wav` file and the `'get_spokenword'` (p.20) function, which wraps around `getsound_vector`.

7.2 Image Analysis

In this section we utilized the `vislabels` function provided for our Lab, built upon a previously created function `acquire_image` (p.18) and created two functions `throw_dice` and `movement` (p.18 and 17).

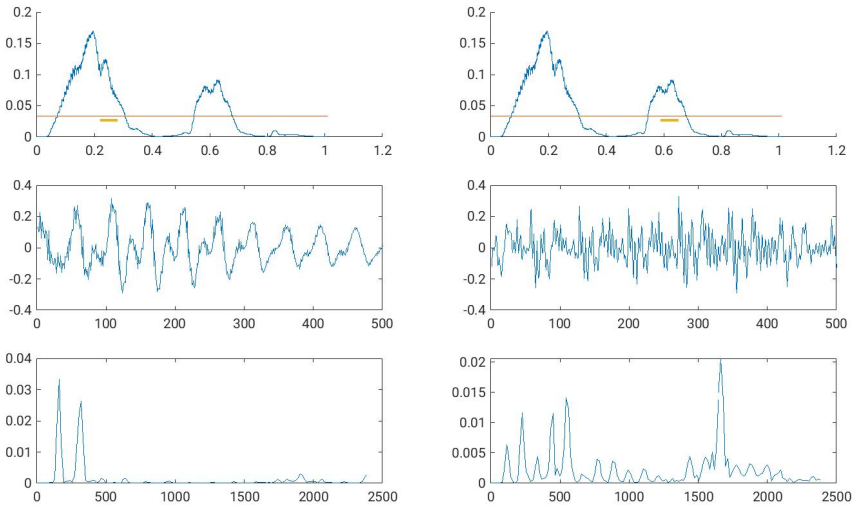


Figure 6: The handling of "Reset"

*The **top panels** are the absolute filtered data of the word "reset". The word is split and where we expect the most the vowel a sample of length 500 is taken (**middle panels**). A fast Fourier analysis of this sample is shown in the **bottom panel**.*

The movement function captures a second image. The gray values of these sequential images are shrunk to a 4×4 image and subtracted from each other. When one of the 16 absolute values is bigger than 10 a movement is detected and when all absolute values are below 2 there is no movement. When the intended movement or rest is reached, an image is taken by the `acquire_image` function.

We tested a couple of setups to catch the dice thrown. Throwing in a box gave some interference of the walls of the box and the dice sometimes rolled out of camera sight. A good setup was found in shaking the dice in a cup and then placing this cup under the camera, as shown in Figure 8 on the following page. We tested the system with a green dice. To make the most of this color we took only the green values of the rgb-image to increase the contrast. After thresholding, opening, complementing and opening the image the BLOBs are counted. Two examples are shown in Figure 9 on page 12.

The good thresholding was very crucial to a correct outcome. Changes in ambient light required changing the threshold value. In our next prototype we should include an automatic threshold leveling that allows for lighting changes.

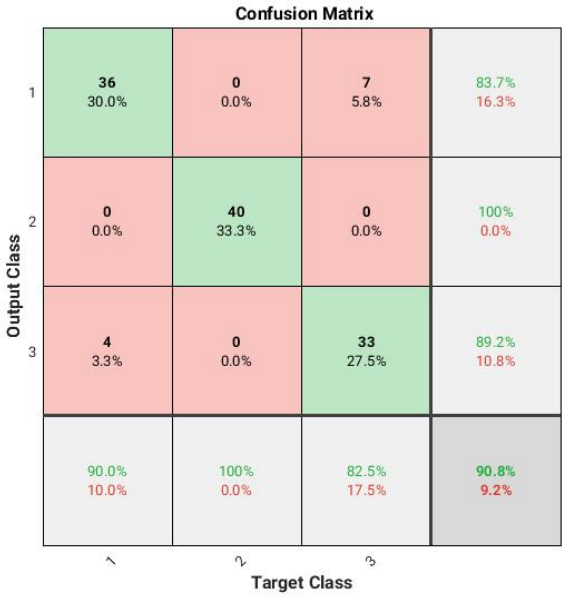


Figure 7: Confusion matrix of the learned neural network
In the matrix 1:'yes', 2:'no' and 3:'rea'

Figure 8: Setup how to throw a dice

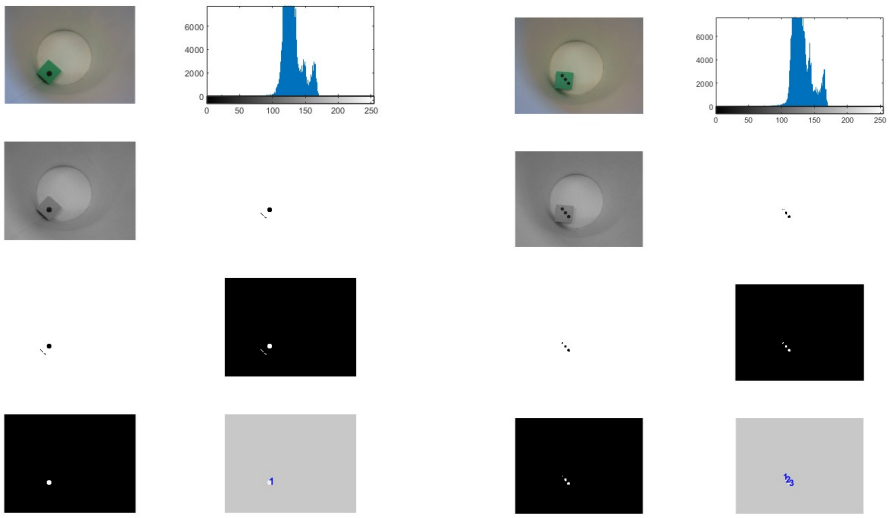


Figure 9: Two dice trows: a 1 and 3

The original rgb-image, the green values as a spectrogram and image, the thresholding to a binary image, the opening, complementing and closing and the BLOBs imaged by `vislabels`*, from left to right and from top to bottom*

7.3 The Game

During development of the game aspect of the system, various parts and functions were tested in order to check and verify the approach was suitable for our needs. The first set of tests were to check a method of displaying the board graphics and also overlay markers that would represent the players. We had to test that our method could handle the various updates that occur, in a clean and elegant manner. This testing consisted of overlay multiple markers over an image and updating the positions in a loop. The results of which were sufficient for us to consider us this approach in Matlab.

The second part that was tested was the function that mapped the players position value to the corresponding marker position on the board. The final function used modulo function to allow the markers to move 'left-to-right' then switch to 'right-to-left' on alternating lines on the board (the function `position` at the end of `play_game2` at p.16). The function was tested by a simple program that looped through [1,100] position values and the display of which was check for accuracy.

The last part of the game that needed to be tested was the game event, which include changes to the players position when they are on either a 'snake' or 'ladder' and also the winning condition that a player has a score above 100. This was tested similar to the stage before, with all position values being looped through and the result of game events were checked for correct behaviour.

The final test for the game module was to put together all the function and test the game by using a random number generator that the computer can play the game itself for multiple players, while we would check for accuracy and desired effects, this can been seen in Figure 10.

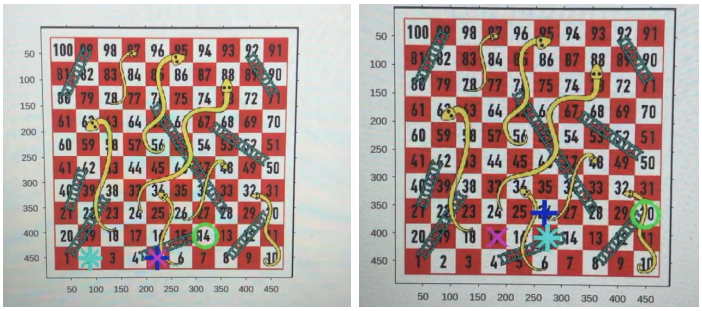


Figure 10: Testing of the game module, taken from one game. Notice overlapping markers.

8 Evaluation

We set out to create a computer version of a classical board game, making it accessible for many people without losing the fun of throwing the dice. We created a game of "Snakes and Ladders" controlled by voice command and spoken directives by the system. Only the physical handling of the dice throw must be executed by the player.

Our system run in a Matlab environment. Although Matlab is a nice program for some mathematical manipulation, image and sound processing and are not its strongest points. Most of the components worked slow but acceptable. The slow processing speed became annoying when all the components were put together. If we had more time for the project we could spent more time optimising the code so that it would preform fast. Else in future versions we recommend to use lower program languages like `C++`.

Also the command recognition was a little bit unreliable. One voice was used for feeding the network with only maximal 120 inputs. Day to day variations in one voice and difference in microphone decreased the accuracy of the command recognition. Improvements in this module are a necessity in future version.

Shaking the dice in a cup did work best. Other dice handling methods, like throwing them in a box, are still interesting to explore.

Overall, we showed that with the aid of the computer, classical board games are still fun to play for modern computer oriented gamers and can be made accessible for less enabled people.

References

(n.d.). https://www.ymimports.com/pages/how-to-play-snakes-and-ladders

Altschul, D. M., & Deary, I. J. (2020). Playing analog games is associated with reduced declines in cognitive function: A 68-year longitudinal cohort study. *The Journals of Gerontology: Series B*, *75*(3), 474–482.

boardgamegeek. (2022). *Games for people with physical disabilities*. https://boardgamegeek.com/thread/597585/games-people-physical-disabilities

Carter, M., Harrop, M., & Gibbs, M. (2014). The roll of the dice in warhammer 40,000. *Transactions of the Digital Games Research Association*, *1*(3).

Nagler, J., & Richter, P. (2008). How random is dice tossing? *Phys. Rev. E*, *78*, 036207.

Silvis-Cividjian, N. (2017). Pervasive computing. *Undergraduate Topics in Computer Science*.

Stein, B. P. (2012). *Dice rolls are not completely random; scientists used new theoretical models and high-speed movies of dice rolls to illustrate*. https://www.insidescience.org/news/dice-rolls-are-not-completely-random

9 Listings

9.1 Main Program

main →

```
clear all % clear all variables
close all % close all figures

% define global camera (we only have to define once)
global cam;
cam = webcam("Microsoft*  LifeCam Cinema(TM)");

% players is a struct of struct player
%   player.name
%   player.f
[players(1).name,players(1).fs]=audioread("sounds/player1.wav");
[players(2).name,players(2).fs]=audioread("sounds/player2.wav");

% main loop
while 1
    %Want_to_play detect movement;
    movement(15);

    %who want to play
    players=input_players(players);
    if isinteger(players), continue; end

    % determine who can start
     [index,~] = who_starts(players);
    %index the players
    players=players(index);

    %play game
    play_game2(players);
end
```

9.2 The Game

play_game2 →

```
function [winnerPlayer] = play_game2(players)
% function to play the game of snakes and ladders for upto 7
% function will for each player take dice roll, move player and display
% results to the screen. This will comntinue untill one player wins (score
% >=100).

% Constants.
    numberOfPlayers = length(players);
    global boardImage;
    global board;
    global step_sound;
    global step_f;


    boardImage = imread('snakes_and_ladder.jpeg');
    win_condition = false; % set win condition no.
    [up_sound,up_f]    =audioread("sounds/up.wav");
    [down_sound,down_f]=audioread("sounds/down.wav");
    [step_sound,step_f]=audioread("sounds/step.wav");

%The ladders and snakes
    p=(1:100);
    p(1)=38;p(4)=14;p(8)=30;p(28)=76;p(21)=42;p(50)=67;p(71)=92;p(80)=99;
    p(32)=10;p(36)=6;p(48)=26;p(62)=18;p(88)=24;p(95)=56;p(97)=78;


% set up
    board=figure;
    current_pos = zeros(1,numberOfPlayers); % create array for positions
    new_pos = 0; % create array for an updated postion

% compute

    while( win_condition ~=  true )
    %disp(["Turn: ", j])
        for i= 1:numberOfPlayers;

        % disp( players(i) )

            diceValue = throw_dice(players(i))
            new_pos = current_pos(i)+diceValue;
            current_pos = display_game(current_pos,i,new_pos);

        % check win
            if new_pos >= 100
                winnerPlayer = players(i);
                current_pos(i) = 100;
                win_condition = true;
```

```
            end % end win if.

            if p(new_pos)>new_pos % ladders
                new_pos=p(new_pos);
                sound(up_sound,up_f);pause(length(up_sound)/up_f);
            elseif p(new_pos) < new_pos %Snakes
                new_pos = p(new_pos);
                sound(down_sound,down_f);pause(length(down_sound)/down_f);
            end % end ladders
        % finish up
            current_pos(i) = new_pos;
            display_game(current_pos);
            pause(0.2);
        end % for player loop end.
     end %for while win end.

    display_position(current_pos);
    pause(0.1)

    disp(["game over. Player ", winnerPlayer, "wins."]);
end


function current_pos = display_game(current_pos,playerIndex,new_pos)
    global boardImage;
    global step_sound;
    global step_f;
    global board;


    figure(board);
    %axis off;
    %hold on;

    if nargin < 2
        playerIndex = 1;
        new_pos = current_pos(playerIndex);
    end

    old_pos = current_pos(playerIndex);
    for pos = old_pos : new_pos
        current_pos(playerIndex)=pos;

        if nargin >1 && pos > old_pos
            sound(step_sound,step_f);
        end

        imshow(boardImage);
        hold on;
        axis off;
        for i = 1:length(current_pos)
            [x,y] = position(current_pos(i));
             plot(x,y, get_marker(i), 'MarkerSize', 25, 'LineWidth',5 );
        end

        pause(0.2)

    end

end

function [x,y]=position(pos)
    q =  floor((pos-1) ./ 10) ;
    even = mod(q, 2);
    odd = mod(q-1, 2);
    y = (453 - q*45);
    x = ( odd*( 42 + 45*( mod(pos-1,10)) ) + even*(453 - 45*( mod(pos-1,10)))  );
end

function marker = get_marker(playerIndex)
    colour="gbcmykr";
    shape="o+x*^v";
    marker=[colour{1}(1+mod(playerIndex-1,7)) shape{1}(1+mod(playerIndex-1,6))];
end
```

9.3 Image Handling

movement →

```
function im=movement(threshold)
% Function MOVEMENT returns if there is a signifivant
% difference in two consecutive images
% if smaller then absolute value of negative threshold -> When movement stops
% or bigger then positive threshold  -> When movement starts
% An image is graped and returned

    im1=getsmallbwimage();
    while 1
        im2=getsmallbwimage();
        rel=abs(im1-im2);
        if threshold < 0
            if min(min(rel))< -threshold
```

```
                im=acquire_image();
                break;
            end
        else
            if max(max(rel)) > threshold
                im=acquire_image();
                break;
            end
        end

        im1=im2;
    end
    acquire_image();
end

function im_out=getsmallbwimage()
    im=acquire_image();
    imshow(rgb2gray(im));
    im_out=imresize(rgb2gray(im),[4,4]);
end
```

acquire_image →

```
function im=acquire_image()
    global cam ;
    %cam = webcam("Microsoft* LifeCam Cinema(TM)");
    %cam = webcam(1);
    im = snapshot(cam);
end
```

throw_dice →

```
function throw=throw_dice(player)
    global pttd;
    global f_pttd;

    if isempty(pttd), [pttd,f_pttd]=audioread("sounds/pttd.wav");  end

    %if isstring(player.name)
    %    fprintf("Throw the dice %s\n",player.name);
    %else
        sound(pttd,f_pttd);
        pause(0.1+length(pttd)/f_pttd);
        sound(player.name,player.fs);
        pause(0.1+length(player.name)/player.fs);
    %end
    %throw=randi([1,6]);

    throw=dice_count();

   while ~ismember(throw,(1:6))
        throw = dice_count();
   end

end


function diceValue=dice_count()
    movement(10);

    im=movement(-2);
    imshow(im)

    im3=im(:,:,2);
    im4=im2bw(im3,0.20);
    im5=imopen(im4,strel('disk',3));
    im6=imcomplement(im5);
    im7=imopen(im6,strel('disk',2));
    [labels,diceValue]=bwlabel(im7);
end
```

9.4 Get Players

input_players →

```
function players = input_players(players)
    global hallo;
    global itaop;
    global f_hallo;

    if isempty(hallo), [hallo,f_hallo]=audioread("sounds/hallo.wav");  end
    if isempty(itaop), [itaop,f_hallo]=audioread("sounds/itaop.wav");  end
```

```
        count=0;
        while 1
            [name,f]=getname();
            if name == 0, players=0; return; end
            if length(name) > 1
                count=count+1;
                players(count).name=name;
                players(count).fs=f;

                sound(hallo,f_hallo);
                pause(0.1+length(hallo)/f_hallo);
                sound(name,f);
                pause(0.1+length(name)/f);
                sound(itaop,f_hallo);
                pause(0.1+length(itaop)/f_hallo);

                answ = yes_no_reset();
                if answ == 2, return; end
                if answ == 3, players=0; return; end
            end
        end
    end
```

getname →

```
    function [name,f]=getname()
        global wiyn;
        global iyn;
        global Fs;

        if isempty(wiyn), [wiyn,Fs]=audioread("sounds/wiyn.wav");  end
        if isempty(iyn), [iyn,Fs]=audioread("sounds/iyn.wav");  end

        while 1
            sound(wiyn,Fs);
            pause(0.1+length(wiyn)/Fs);

            [name,f]=get_spokenword("Your name",3);

            sound(iyn,Fs);
            pause(0.1+length(iyn)/Fs);

            sound(name,f);
            pause(0.1+length(name)/f);

            answ = yes_no_reset();
            if answ == 1, return; end
            if answ == 3, name=0; return; end
        end

    end
```

who_starts →

```
    function [index,count]=who_starts(players, index, count)
        l=length(players);
        % first call ?
        if nargin < 2, index=zeros(l,1);count=1; end

        % get throws for this round
        d=zeros(l,1);
        for i=1:l
            if index(i)==0, d(i)=throw_dice(players(i));end
        end
        %d'

        % select single values
        for i=1:6,if length(d(d==i))==1, index(d==i)=1;end;end
        index(index==0)=-1;

        for i=6:-1:1
            ld=length(d(d==i));
            if ld==1 % single value is okay
                index(d==i)=count;
                count=count+1;
            elseif ld>1 % thows are the same, select and throw again
                index(d==i)=0;
                [index,count] = who_starts(players,index,count);
            end
        end
        %index'
    end
```

9.5 Voice Recognition

getsound_vector →

```
function [wave,Fs]=getsound_vector(time,name,debug)
% Function getsound_vector generates a wave sample of maximal time seconds
% If no sound it returns (0)
    if nargin < 2, debug=0; end
    if nargin > 1 && isstring(name)
        if 0 < exist(name)
            [wave,Fs] = audioread(name);
            return;
        end
    end

    Fs=8000;
    rec=audiorecorder(Fs,16,1);
    recordblocking(rec,time);
    sound=getaudiodata(rec);
    Fs=rec.SampleRate;% get correct sample rate when Matlab-online

    s1=movmean(abs(sound),ceil(Fs/50));
    bg=mean(s1(1:Fs/4));
    ms=bg*2;
    L=length(s1);

    if debug > 0.1
        x=(0:L-1)/Fs;
        figure;
        subplot(1,2,1);
        plot(x,sound);
        subplot(1,2,2);
        hold on;
        plot(x,s1);
        plot([0,L/Fs],[ms,ms]);
        hold off;
    end

    j=1;
    k=0;
    s=zeros(L,1);
    lim=Fs*0.25;

    for i=1:L

        if s1(i)>ms % sound
            s(j)=sound(i);
            j=j+1;
        else % no sound
            if j > lim % we are in a word
                k=k+1;
                j=j+1;
                if k > lim % end of word
                    wave=s(1:j-1);
                    if nargin > 1 && isstring(name)
                        audiowrite(name,wave,Fs);
                    end
                    return;
                end
            end
        end
    end
    wave=(0);
end
```

---

get_spokenword →

```
function [wave,fs]=get_spokenword(word,time,debug)
    if nargin < 3, debug=0; end
    if nargin < 2, time=3; end

    fprintf("\nYou have %d sec. tp pronounce the word '%s' ...",time,word);
    [wave,fs]=getsound_vector(time,0,debug);
end
```

---

getpeaks →

```
function res = getpeaks(word,f,debug)
    if nargin <3, debug=0; end
    res=[];
    Ls=500;
    Npks=10;

    s1=movmean(abs(word),ceil(f/50));
    co=0.2*max(s1);
    L=length(s1);
```

```
    X=(0:149)*f/Ls;

    t=0;
    l=0;
    for i=1:L
        if s1(i) > co
            t=t+1;
        else
            if t>0
                if t>700
                    p=ceil(i-700);

                    y=word(p:p+Ls-1);
                    Y = fft(hamming(Ls) .* y);
                    P2 = abs(Y/Ls);

                    if debug
                        figure;

                        subplot(3,1,1);
                        hold on;
                        plot((0:L-1)/f,s1);
                        plot([0,L/f],[co,co]);
                        plot([p/f,(p+499)/f],[0.8*co,0.8*co],'LineWidth',2);
                        hold off;

                        subplot(3,1,2);
                        plot(y)

                        subplot(3,1,3);
                        plot(X,P2(1:150));
                    end
                    l=l+1;
                    [pks,locs]=findpeaks(P2(1:150),X(1:150),'SortStr','descend');
                    res(l,:)=[pks(1:Npks)',0.001*locs(1:Npks)];

                end
                t=0;
            end
        end
    end
end
```

---

network/traindatain →

```
words=["YES","NO","REA"];
M=40;
res=zeros(M,21);

for i=1:M
    rn=randi([1 3],1,1);
    [w,f]=get_spokenword(words(rn),2);
    pks=getpeaks(w,f);
    if length(pks) > 0, res(i,:)=[rn, pks];end
end
save('learn2','res')
```

---

network/learn →

```
words=["YES","NO","REA"];

data = load('learn3.mat');
labels=data.res(:,1)';
inputs =data.res(:,2:end)';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
data = load('learn2.mat');
[l,]=size(data.res);

labels= [labels, data.res(:,1)'];
inputs= [inputs, data.res(:,2:end)'];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
data = load('learn1.mat');
[l,]=size(data.res);

labels= [labels, data.res(:,1)'];
inputs= [inputs, data.res(:,2:end)'];

%labels(labels==4)=1;
%labels(labels>4)=4;


targets=dummyvar(labels); % make the matrix target
targets=targets';

net = patternnet(10);
[net, ~] = train(net, inputs, targets);
predicted= net(inputs);
```

```
[c,~,~,~] = confusion(targets,predicted);
plotconfusion(targets,predicted)
save("myvoice.mat","net");
```

9.6 Spoken Text

sounds/make_spoken →

```
[w,f]=get_spokenword("Say Yes,No or Reset",4);
sound(w,f);
audiowrite("synr.wav",w,f);
pause(4);

[w,f]=../get_spokenword("What is your name?",4);
sound(w,f);
audiowrite("wiyn.wav",w,f);
pause(4);

[w,f]=get_spokenword("Is your name ",4);
sound(w,f);
audiowrite("iyn.wav",w,f);
pause(4);

[w,f]=get_spokenword("please throw the dice",4);
sound(w,f);
audiowrite("pttd.wav",w,f);

[w,f]=get_spokenword("Hallo",4);
sound(w,f);
audiowrite("hallo.wav",w,f);
pause(4);

[w,f]=get_spokenword("Is there an other person",4);
sound(w,f);
audiowrite("itaop.wav",w,f);
pause(4)

[w,f]=get_spokenword("player one",4);
sound(w,f);
audiowrite("player1.wav",w,f);

[w,f]=get_spokenword("player two",4);
sound(w,f);
audiowrite("player2.wav",w,f);
```