Do Video Game Players and Non-Video Game Players have different Visual Cognitive Skills?

Julian Steinke

University of Twente

Bachelor thesis

First supervisor: Rob van der Lubbe

Second supervisor: Suzanne Vosslamber

Abstract

Action video games have a great popularity for entertainment purposes. Recent studies suggest that these games might also offer a chance to train cognitive skills, such as logic, planning and visual skills. This study will try to compare the visual cognitive skills of video game player and non-video game players. The executive control, visual working memory and temporal and visuospatial attention of students with experience in the action video game *Counter Strike* were compared with students without experience in this game. The participants were tested with a spatial Stroop task, an attentional blink paradigm, a change detection task and a useful field of view task. The results suggest no difference between the two groups. This discrepancy with the literature addressed above might be due to the young age of the participants, the too little experience in the game of the gaming group. Other literature supports these claims.

Contents

Introduction

Since their emergence in the 1970s, video games gained a great popularity. This popularity created great debate between supporters and opposers of videogames. Supporters focus on the relaxing aspects and that video games enhance cognitive skills (Van Hattum, 2017). Opposers focus on negative negative aspects. Especially games with violent content got public attention for their possible effect of increasing violent behaviour. As Ferguson (2007) points out, results on this matter are mixed. Other research also suggests an increase in the symptoms of ADHD when video games are played for a longer time (Chan & Rabinowitz, 2006). A second factor is the physical inactivity when playing. Long time playing of video games is linked to a lower bone mineral density (Shao, et al., 2015) and obesity in young adults (Marshall, Biddle, Gorely, Cameron, & Murdey, 2004).

Recent research suggests that action video games are not just a form of entertainment, but also offer the possibility for the training of cognitive skills. Studies suggest a positive effect of playing video games for various mental processes in for example fields of logic, strategic planning, visual attention and working memory (Boot, Kramer, Simons, Fabiani, & Gratton, 2008; Basak, Boot, Voss & Kramer, 2008). Action video games require players to use their cognitive skills very fast and accurate to play a game and to be successful in it. Therefore, it seems obvious that cognitive skills could be trained with playing these games.

However, other studies question these findings. It was also found that practice with action games does not improve the performance on perceptual tasks any more than other types of video games or no gaming at all (Van Ravenzwaaij, Boekel, Forstmann, Ratcliff, & Wagenmakers, 2014). A meta-analysis by Powers, Brooks, Aldrich, Palladino and Alfieri (2013) shows that the effect size of training in action games differs strongly between quasi-experiments that produce large effect sizes and true experiments almost no effect. This result suggests, that the proclaimed cognitive advantages may not be gained by playing video games, but may be already well developed and thus are the reason to play. Therefore, a self-selection bias could be the reason (Heckman, 1990).

Choice of a specific game

Since the research will be conducted in the Netherlands, it is important to keep the Dutch market in sight in order to find a fitting game. According to the gaming platform *Steam*, the most played action game in the Netherlands is the first person shooter *Counter-Strike: Global Offensive* (CS:GO) by Valve and Hiddenpath Entertainment (2012) with approximately 713.000 players (Steamspy, 2017). CS:GO is well suited for research in the action game field not just because it has a big player base and therefore a big video-gaming population. It also features the core elements of the genre. The player can choose from multiple competitive online game modes featuring different combat scenarios against other players. Secondly, the player can choose from different weapons to operate, but most vehicles or classes with special abilities. Other games of the genre with a great player base as *Battlefield 1* (EA DICE, 2016) for example features different classes with different specifications and vehicles giving some players an advantage over others in a one to one situation. Lastly, CS:GO offers a ranking system that is based on the success and skill of the player and not the amount of time played. The distribution of the 18 ranks of the game under the players approaches a normal distribution (Gamurs Pty Ltd., 2017). Thus, it is possible to tell which player is more experienced and scores higher in the game on average.

Action video games and cognitive skills

Visual cognitive skills are especially important when playing an action video game as CS:GO. In this section, the choice of the specific skills for this study is explained by linking them to specific elements of the game. It is important to note, that these skills are almost never used alone but together.

One of the skills needed for a successful player of CS:GO is executive control (Strobach, Frensch, & Schubert, 2012). For action games, this skill includes the ability to identify a stimulus correct, even if some information is contradictory. Secondly, it includes the very important skill of fast reaction to these target stimuli. The Stroop task is mostly used to test this skill. This requires the participants to use their executive control fully because they have to plan their possible reaction to a stimulus, have to deal with a possible conflict between the characteristics of the stimulus and have to inhibit motivation to react to the unimportant stimulus. A spatial variant of the Stroop task is chosen for this study, thus with a directional stimulus, like an arrow, and the position of this stimulus that can be congruent or not (Liu, Banich, Jacobson, & Tanabe, 2004). The position of the arrow is the target characteristic that the player has to react to. This can be compared to a key element of Counter Strike, the attention to the position of players of the own team. Especially in stressful situations in the game, it is easy to confuse players of the own and the other team. Whether the player spotted belongs to the own team or not is shown by the colour of the name shown above the figure. The appearance of the figure in the game has to be ignored and the colour of the name above the seen figure is important for decision making, thus an engagement or not.

A second important skill identified for action games like CS:GO is visual working memory (Blacker, Curby, Klobusicky, & Chein, 2014). This skill is needed to remember multiple visual cues and recognise a possible change. This is tested with a change detection task, wherein multiple objects have to be remembered for a short time and the participant has to recognise a possible change when the objects are shown again. A comparable change detection task was used by Jones and Berryhill (2012). This skill is needed in action video games because a player has to identify clues about the position of the other team. The player often has to remember details of his surroundings in a short amount of time. A change in these surroundings might be caused by the enemy and thus, give an indication about the other players. An example could be destroyed blockades, fire, bullet holes or opened doors that can give away the position of the opposed players.

Another skill thought to be important for CS:GO is visuospatial attention (Green & Bavelier, 2007). More specifically, this skill describes the ability to spot a visual cue in the whole field of view. This skill can be tested with a useful field of view test. This test requires the detection of a change that can occur in the central or peripheral vision. The peripheral vision is very important in games like CS:GO, because the player has to look out for his opponents and thus has to be alert to detect certain cues, like the model of an enemy player or a thrown grenade anywhere on the screen.

In addition to the spatial dimension of visual attention, the temporal dimension is also important. This skill requires to recognise and memorise important information, even if it is shown very briefly. The attentional blink can be used as an indicator for this skill (Green & Bavelier, 2003). It uses two targets that are shown very briefly with different non-target stimuli between them. The blink describes the failure to memorise the second target when the lag between the targets decreases, thus when the detection or processing of the second target falls into the refractory period of the processing of the first target (Raymond, Shapiro, & Arnell, 1992). This effect can be a problem when multiple important stimuli are visible for the player very briefly, like two players of the opposing team appearing for a very short time and very fast after another. Secondly, the player always has to look out for certain cues when also many other, but unimportant cues like parts of the environment are presented.

The main research question of this study is if players and non-players of CS:GO have different visual cognitive skills. Based on the four skills chosen, the hypotheses for this study are the following. Video game players and non-video game players perform differently in a Stroop task. The same is thought of the attentional blink, change detection task and their usable field of view.

Because of the ranking system, it is possible to compare the performance within the video game playing group. The hypotheses for this comparison are, that players of different ranks in CS:GO perform differently on the Stroop task, the attentional blink paradigm, the change detection task and their useful field of view.

Methods

Participants

All participant were male students of the University of Twente (*N*=36). The mean age was 22 with the youngest being 18 years old and the oldest having an age of 29 (*SD*=2.476). 15 Participants were Dutch, 17 German and 4 of another origin. No one had an impaired visual acuity limiting the capability of performing in the tests which was measured with a Snellen test that is described later on. Two of the participants reported having weak colour vision, one limited on red and one limited on green. Of the 36 participants 24 reported having experience in Counter Strike, 12 did not. Of the gaming group, the ranks had a mean of M=7.957 (SD=7.289). The participants were rewarded with points for their own study when participating. It has to be noted, that the participation in studies like this is mandatory for them, the choice of studies, however, is completely free. Only men with either experience in Counter Strike or none at all could participate.

Apparatus and Materials

This section has two categories, the tasks and the questionnaire used. The experiment itself was conducted with a computer on which the programs ran that are described in the next section. The two computers used ran the programs with Spyder 3.1.2 and Spyder 3.5. The first computer used a setup with an Intel Core i5-4200M CPU at 2.5GHz, 4 GB RAM and a Nvidia GeForce GTX 760m graphics card. The OS used on this PC was Microsoft Windows 8 64-Bit. The second computer used a setup of an Intel Core i3-6100U CPU at 2.30GHz, 4 GB RAM and ran Microsoft Windows 10 64-Bit.

Tasks. In every experiment, a welcome screen explained the basic instructions. The participant has to start the next trial by pressing the spacebar. The program scripts can be found in Appendix A. The programmes were written in Python 3.5.2. All programs used grey backgrounds except for the change detection task, where a white background was used because of grey stimuli. Also, the change detection task was the only task not using a 1080x1920 resolution but 1200x700. Every program logged the variables needed in a .txt file that could later directly be imported in SPSS.

Stroop task. The used variant of the Stroop task included arrows as stimuli. These arrows appeared in the upper or lower half of the screen with an either congruent or incongruent orientation (i.e. pointing up or down). The arrows were always presented on the middle axis of the screen in the same position. Each trial was started immediately after the participant pressed the spacebar. The participant would have to indicate the position of the arrow with the arrow keys UP for an arrow positioned in the upper half of the screen and vice versa. The position of the arrow is the relevant feature and the direction is the irrelevant feature. After the response, the reaction time and if the response was correct or not was shown on the next screen. The participant then had to start the next trial. This task had 60 trials. Half of the trials were congruent and also half of the trials had the arrows appear in either half of the screen (*Figure 1*). The logged variables of the Stroop task were the reaction time, the correctness of the input and whether the trial was incongruent or not. After the spacebar was pressed by the participant to start the trial, the arrow was shown and the clock to log the reaction time was started.

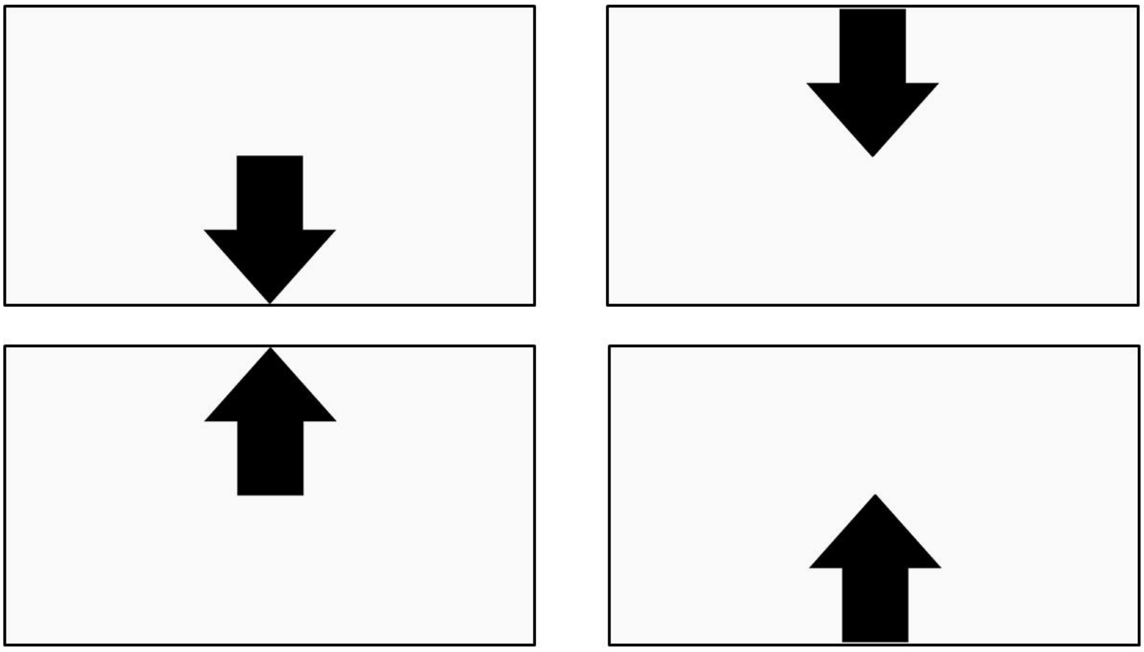


Figure 1. The four possible screens of the Stroop task. Congruent trials are shown on the left, incongruent on the right.

Attentional blink. After the start of each trial, a row of 31 characters was shown to the participant, every character appeared for 33ms with an interstimulus interval of 50ms. The stimuli were 1,4cm tall (capital letter height), all letters shown were lower case letters. One letter was white and one character could be a number that appeared between one and eight characters after the white letter. 50% of the trials had a number. The white letter, its position and whether or not the trial had a number was random. If a number was included, it was between one and eight characters after the white letter. The screens of this task are shown in figure 2, the characters were not presented in the middle of the screen, but on the position where the 'Press space' on the screen before was located. If the stimuli were presented at another location, it would have been possible to miss the first target. After the characters were shown, the participant would have to press the key with the letter that was white and then Y if a number was shown and N if not. Both responses were asked on a different screen. Immediately after both responses were received, a new screen indicated whether the responses were correct or not. This task had 40 trials. After the last trial, the program logged the correctness of both inputs and the position of the number in relation to the white letter.

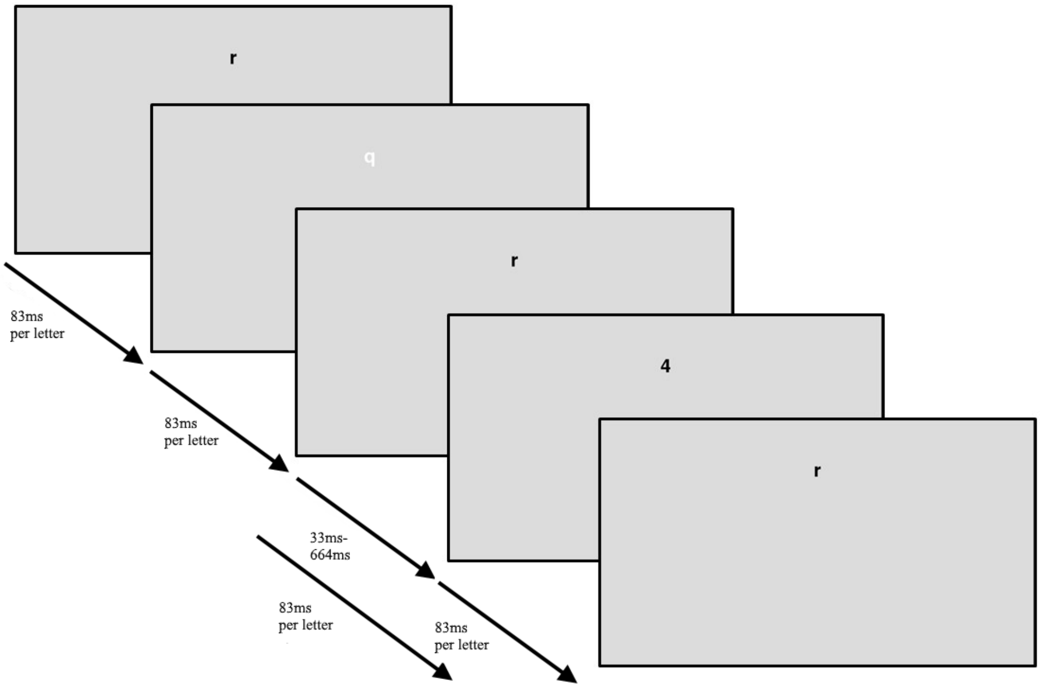


Figure 2. Stimuli of the attentional blink task. The letter ‘r’ represents a random amount of random black letters. The number did only appear in half of the trials.

Change detection task. Five rectangles in a random position and random colour were shown for one second. Immediately after that, a masking screen appeared, also for one second, wherein the whole screen was covered with coloured rectangles in order to cope with a possible effect of an afterimage. Then, a single rectangle was shown for one second and a second mask. This sequence of stimuli is shown in Figure 3. On 50% of the trials, the rectangle in the second sequence was also shown in the first sequence. As soon as the second mask was shown, the participant had to indicate whether the single rectangle was also present in the first sequence of five in regards to position and colour with Y for yes and N for no. Immediately after the response had been given, a screen indicated whether the response was correct or not. This task had 50 trials. The colours used were black, grey, red, green, blue and yellow. The rectangles could appear on a grid of 5x5 with each rectangle covering a 25th of the screen. Because of the grid being defined in parts of the screen, this grid is independent of the solution of the screen. The program logged the correctness of the input and whether or not one of the rectangles was in both sequences.

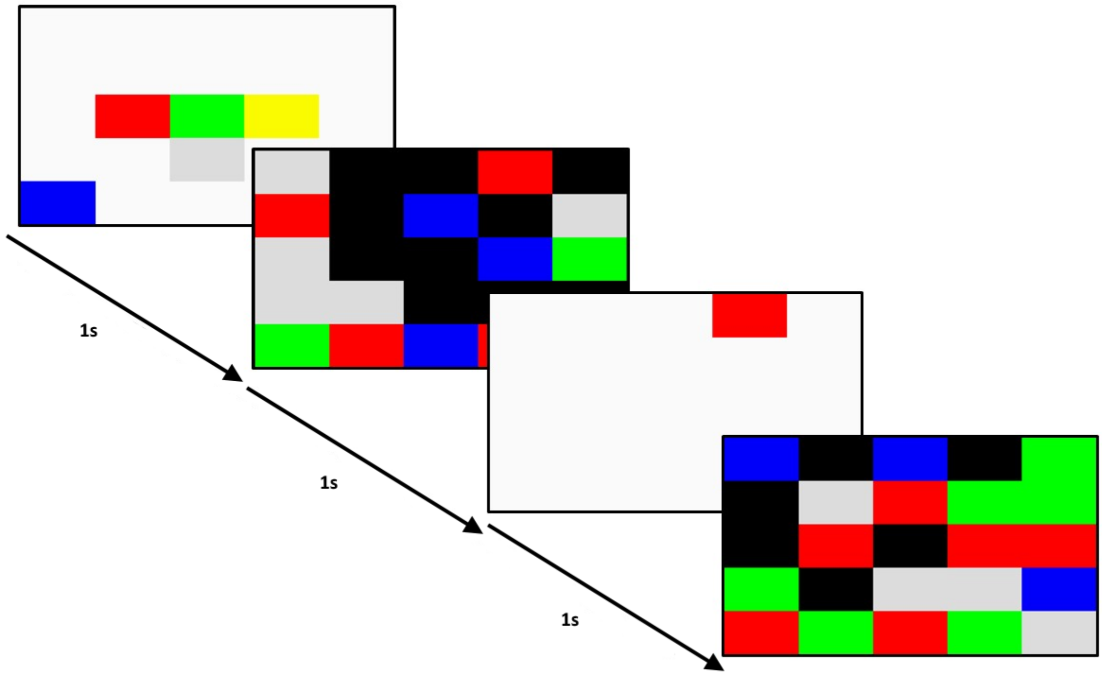


Figure 3. The different states per trial of the change recognition task. Each state was shown for one second. One-half of the trials showed a rectangle on the third screen that was also in the first.

Useful field of view. Black circles on four axes, vertical, horizontal and the two diagonals, were shown to the participants distributed on 80° of their field of view, he or she had to focus on the centre. The distance to the screen was fixed to assure the circles were on the correct degrees. One of the circles would turn red and be visible for 200ms. Then, a masking stimulus was shown, with 50 red and black circles in random positions for one second, also to cope with possible afterimages (Figure 4). The first screen then appeared again and the participant had to click on the circle that turned red with the mouse. This task had 60 trials. The grid for the circles was divided into eights of the screen, thus the circles were on the 0°, 10°, 20°, 30° and 40° positions of each side of the participants' eyes. Because of the four axes used, there were 25 circles. The program logged the correctness of the input and the position of the red circle. The contrast between the red colour of the target and the black of the other circles and the size of the circles were chosen in such a way, that participants with protanopia had no problems with spotting the target.

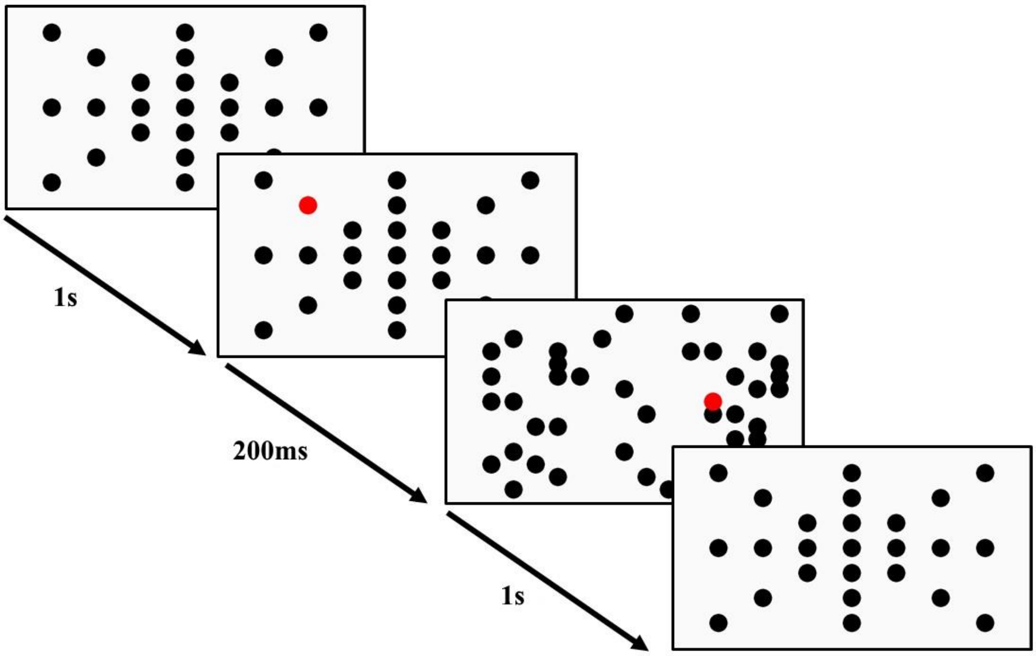


Figure 4. One trial of the useful FOV task with the times between the different states.

Questionnaire. The questionnaire (Appendix B) given to the participants was used to get the demographic information of the participants. It included their age and nationality and their experience in CS:GO. Additional information asked for was any kind of colour-blindness, because of the coloured stimuli used and the preferred hand.

A visual acuity chart was used in order to test if the participant had an inaccurate vision. The chart was printed in A4 and the participants stood 2,8m away from it. The chart can be found in Appendix C.

Design & Procedure

For this study, a between-groups design was employed. The experiment was conducted in a controlled environment without disturbances. The participants were told the purpose of the study and what tasks they would have to perform. After they signed an informed consent (Appendix D), they were given the instructions for the experiments in printed form. These instructions were next to the computer in printed form the whole time. The participant had to perform all four tests in five practice trials each in order to assure that the instructions had been understood. If the participant had no more questions, the real experiment started. The order of tasks was always the same.

After the experiment, the participant filled in the questionnaire and took the visual acuity test with both eyes separately. The participant was then debriefed that he or she was not manipulated in any way and that the true purpose of the study was fully disclosed. The participant was thanked for the participation and left his or her email address when he or she was interested in the results of the study. The participant was thanked for the participation and left his or her email address when he or she was interested in the results of the study. The whole session took 30 minutes, the experiments themselves 20 minutes.

Statistical Procedures

The .txt output given by the program was converted into SPSS. The mean scores for all variables per participant were calculated. For the Stroop task, a one-sample *t*-test was used to ensure that the Stroop effect was present. The same was done for the useful field of view task with a Wilcoxon test. The presence of an attentional blink was tested with a Friedman test. For all results, a Kolmogorov-Smirnov test was conducted to test for a normal distribution. If the results would have been distributed normally, a t-test for independent variables grouped on the experience of the participants could have been conducted. Otherwise, a Whitney-Mann U test would have been conducted with the same grouping variables. All tests were also used with the used computer as a grouping variable in order to look for a categorical difference between the results found on the different computers.

The Stroop effect per participant was calculated as the difference of the mean time needed to react to a congruent stimulus and the time needed to react to an incongruent stimulus. This returns the average delay between the two states in milliseconds.

The level of significance for all tests was chosen to be α=0.05.

For the analysis of the attentional blink task, all trials with a correct answer on both targets were selected. These resulted in values between 1 and 8 for steps of 83ms (50ms between the characters and 33ms per character). Thus, these values were transformed to the related time, ranging between 83ms and 747ms.

For the useful field of view task, the mean score of correct responses only in the periphery was calculated. This returns a proportion value that was transformed with an arcsine transformation.

Because of the rank system of the game, it is also possible to compare the participants within the gaming group with their ranks as a subgroup. All tested variables remain numeric, but because the rank system has 17 possible ranks, the test for normally distributed values will be a one-way ANOVA or a Kruskal-Wallis test if the values are not distributed normally.

Results

The mean score on the Stroop task, thus the mean difference between the reaction time on congruent and incongruent trials was M=28.2ms (SD=47.8ms). A Kolmogorov-Smirnov test was conducted in order to test the distribution and indicated a normal distribution. Figure 5 shows the results of the Stroop effect. The t-test performed to assess the Stroop effect showed a significant difference between the results of congruent and incongruent trials (p=.001, $t(35)$=-3.531), confirming the presence of the Stroop effect. A performed t-test showed no significant difference between the two groups in their performance in the Stroop task (p=0.670, $t(34)$=-0.429). The null hypothesis has to be retained. The one-way ANOVA performed for the differences between ranks also returned a not significant result (p=0.963, $F(11)$=0.323) indicating no difference in performance on the Stroop task between players of different ranks.

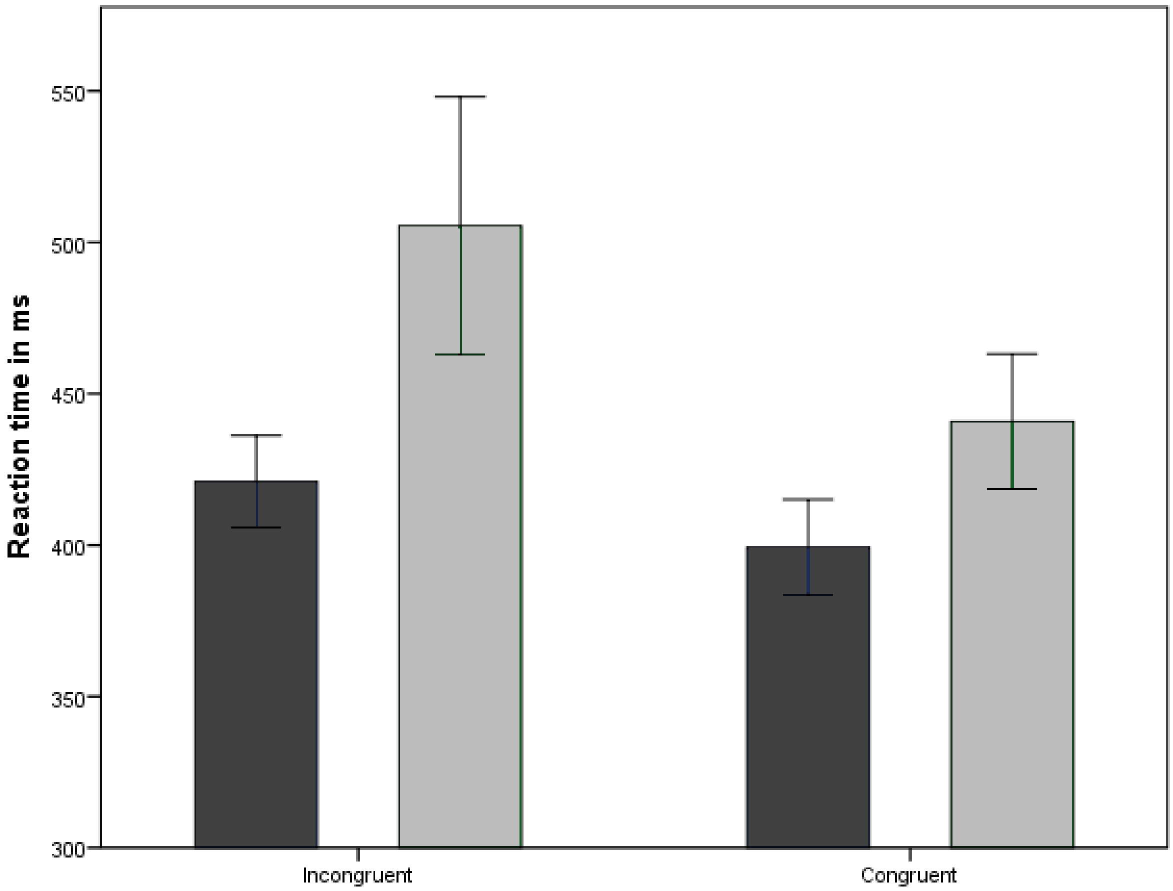


Figure 5. Mean reaction time for congruent and incongruent trials of the Stroop task. The dark bars represent the group with experience in CS:GO.

The attentional blink task shows a mean score of *M*=74% (*SD*=44%) for a correct identification of the second target when the first target. The Kolmogorov-Smirnov test indicated no normal distribution, but because of the design of the task, this had to be expected. The attentional blink was present according to the Friedman test (p=0.000, X^2=1083.000, α=0.05). The graph (Figure 6) also clearly visualises the blink as a decline between 200ms and 400ms. A Mann-Whitney U test was performed to compare the results of the blink, the test showed no significant difference as shown in Table 1 (p=0.309, Z=-1.018), indicating no difference between the two groups in their attentional blink. The overall results, including the average correct scores on both targets per participant, were distributed normally.

To control for a categorical difference, a t-test was performed for mean scores. This t-test indicated a significant difference between the two groups (p=0.039, t(35)=1.824). A performed Kruskal-Wallis test (Table 2) to compare the attentional blink between the different ranks showed no significant difference between the ranks (p=0.357, Z=-3.531).

Table 1. *The result of the Mann-Whitney U test for the attentional blink grouped on non-video game players and players with experience in CS:GO.*

Mann-Whitney U	62338.50
Wilcoxon W	88444.50
Z	-1.02
p-Value	.31

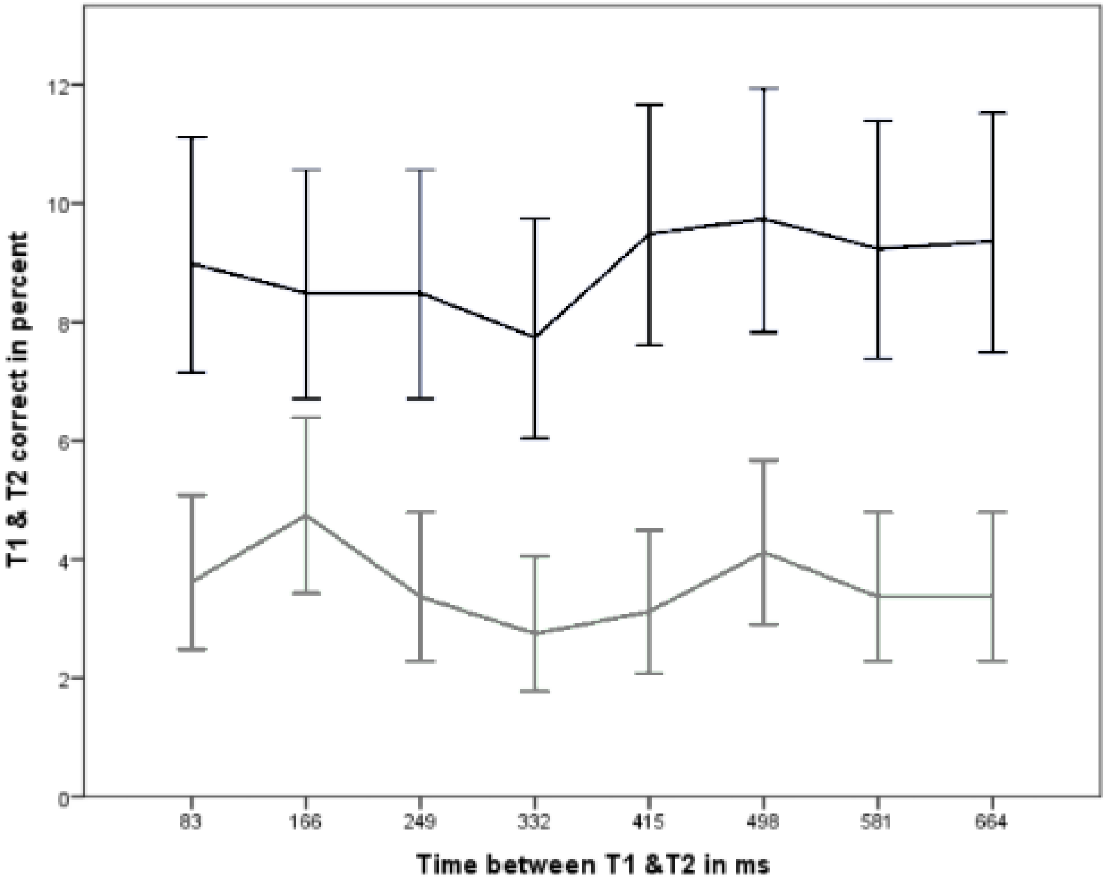


Figure 6. Percentages of the correct identification of target 1 (T1) and target 2 (T2) time between T1 and T2. The line above represents the group with gaming experience.

The change detection test showed a mean score of *M*=75.6% accuracy with a standard deviation of *SD*=13.6%. Figure 7 shows the scores per group. The performed Kolmogorov-Smirnov test indicated a normal distribution. The performed t-test showed no significance (p=0.852, $t(34)$=0.188) indicating no difference between the groups. One-way ANOVA was performed in order to look into the difference in scores between different ranks of the playing group, this turned out not being significant (p=0.958, $F(11)$=.336).

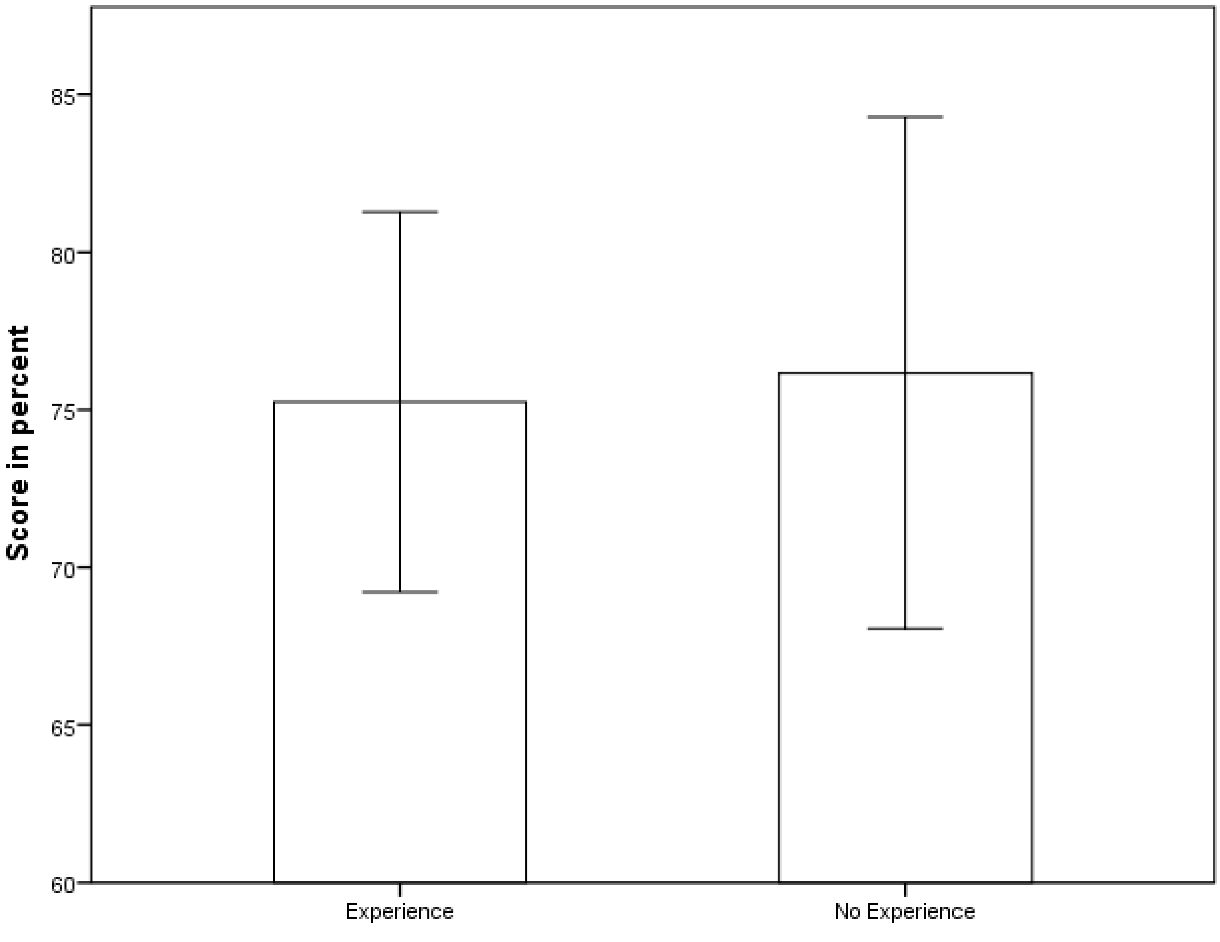


Figure 7. *Correct scores on the change detection task in regarding the experience in CS:GO.*

The useful field of view test showed a mean score of *M*=0.912 (*SD*=0.048), the performed Kolmogorov-Smirnov test indicated a not normal distribution. Thus a Mann-Whitney-U test was performed for this variable. This showed no significant difference (p=0.370, Z=-0.897), indicating no difference between the two groups. A Kruskal-Wallis test was performed to analyse the difference in scores on the useful FOV task grouped on ranks of players (Table 2). This difference between ranks turned out to be not significant (p=0.422, $X^2(11)$=11.257).

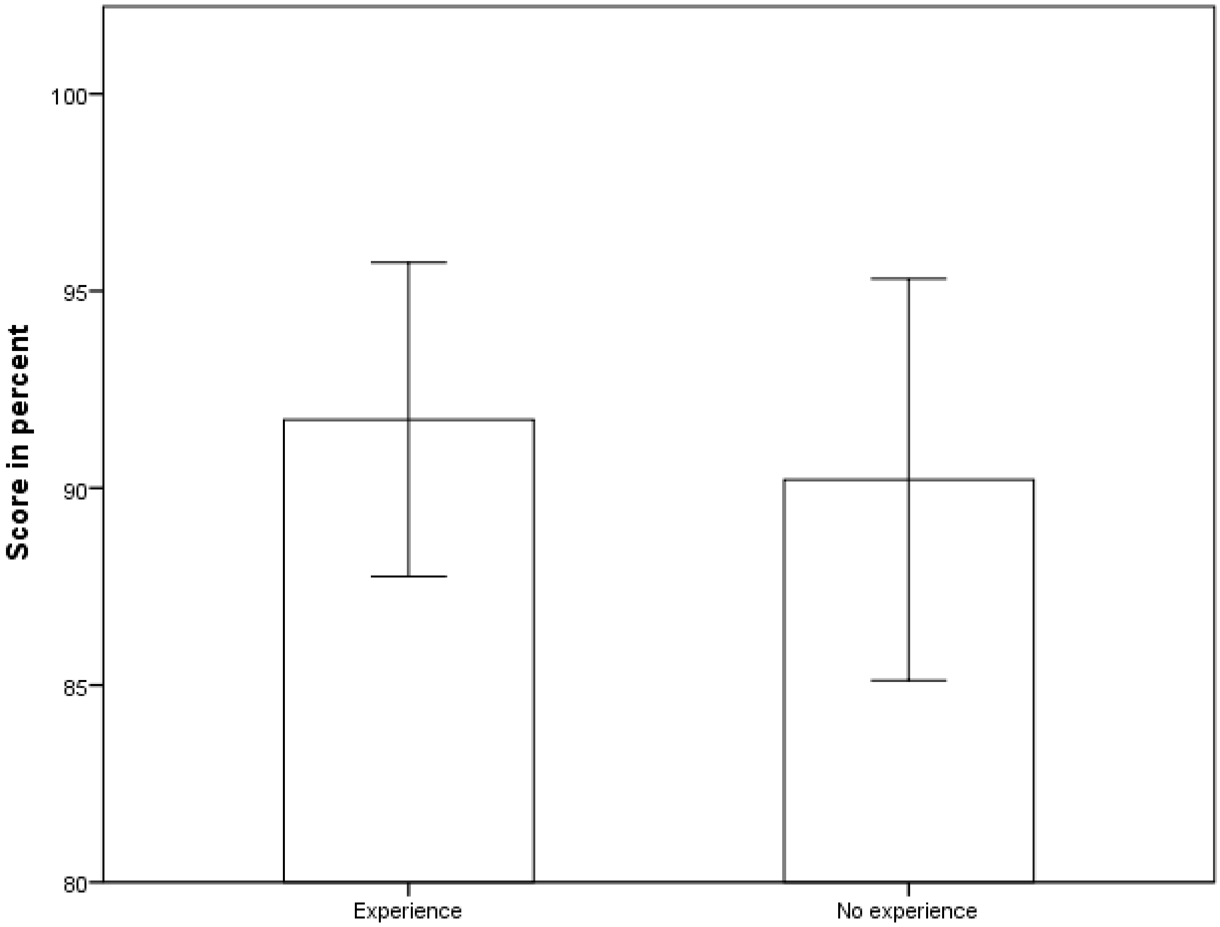


Figure 8. *Scores on the useful field of view task for both groups.*

Table 2. *Results of the Kruskal-Wallis tests for the attentional blink, the change detection task and the useful field of view task, all grouped on the ranks of the players.*

	Attentional blink	Change Detection	Useful FOV	Attent. Blink Mean
Chi-Square	11.00	5.99	11.26	1486
df	10	11	11	11
p-Value	.36	.87	.42	.19

Discussion

The main research question of this study was if video game players and non-video game players have different visual cognitive skills and, if so, how big these differences are. In short, the results presented here indicate, that there is no difference in the visual cognition of the two groups.

A difference in executive control and visual working memory is very unlikely, even with a larger sample size.

Video game players and non-video game players have no difference in their executive control and their visual working memory. This means also that it is unlikely, that these two skills can be trained with action video games.

The used attentional blink paradigm showed no difference between the two groups for the blink itself, but the video game players performed better, meaning they were abler to correctly identify both targets. The blink, however, affected them in the same way as the non-video game playing group. Thus, they are more attentive in general but have a similar refractory period to process the first stimulus. This effect can be explained with the experience in action games because the players have to expect opponents all the time. It seems likely, that this can be trained with action video games because other forms of computer-based training of attention seem also to b effective (Steiner, Sheldrick, Gotthelf, & Perrin, 2011).

Video game players and non-video game players also seem to have no differences in visuospatial attention and the results suggest a great certainty in this. However, this result could be due to the task itself because it appeared to be too easy. It is reasonable to talk of a ceiling effect. There is also no difference found within the video-game playing group based on the ranks for any of the tasks. But these results have to be interpreted carefully because many of the ranks had very few samples, some just one. The representative value of these results has to be questioned.

One answer to the question, why these results differ from those found by comparable studies may be the age of the participants. This study only used young adults with basically good health. Many studies supporting cognition training with video games use older adults as a target group and for rehabilitation purposes (Barry, Galna, & Rochester, 2014 & Belchior, et al., 2013 & Ogawa, You, & Leveille, 2016). It can be expected, that this group shows a bigger difference between people with and without training in video games than young adults that have better visual cognition skills on average. The results found by this study and the study above are therefore not contradictory because this study has a different target group.

Another possible explanation for the indifference between the group is that the expert group had not enough real experts. Boot et al. (2008) suggested the same as an explanation for comparable results because gaming experts in their study performed significantly better than non-players. And indeed, the video game playing group of this study included players with low playtime and also without a rank in the game. It may be possible, that these players did not have enough experience in the game to influence their cognition.

It is also important to note, that many comparable studies train people inexperienced in video games with a target game and another different game for a control group to cope with a possible self-selection bias. The exclusion of such a bias is not possible in the design chosen here, this is only a comparison of video game players and non-video game players. It is not possible to draw any conclusions on effects on visual cognition caused by action video games.

The age of the participants is a strong point of this study because young adults are a big group in video gaming and most action games, especially with violent content are intended for this group. Most research in this area focusses on older adults and children (Belchior, et al., 2013). CS:GO as a target game offers a great opportunity for this type of study. It is popular, basic for its genre and offers the possibility of identifying and comparing players of different experience. Also, the tests used for the test battery measure the correct cognitive factor and can measure differences well. This test battery can thus be used for this type of study and for other cognition studies. Especially the change detection task and the spatial Stroop task can be reused without any changes.

Recommendations

The task battery used in this study was also used to compare results within the group of gaming participants, but a greater sample for this subgroup is needed in order to draw any conclusions about a possible effect of a difference in expertise between players on cognition. The useful field of view task used for this study needs to be improved. The targets have been correctly presented in the periphery but were too easy to spot apparently. It is suggested to shorten the time the target stimulus is presented, or to make the target's appearance less obvious by, for example choosing a less striking change in colour that could still be spotted by a colourblind. This study could be performed with professional players as the gaming group. This would exclude the possibility of including players with almost no experience and would thus be an opportunity to compare population with the highest expertise with inexperienced players. If the results of this study are reproduced in such a study, a difference in visual cognitive skills between the two groups would be very implausible.

All tests used here show direct effects, but it would also be important to look into long-term effects with a vigilance test for example. Also, completely different tests should be done, measuring other aspects of cognition. To stay with CS:GO, hearing is very important also.

Conclusion

The conclusion has to be made, that there is no difference in the visual cognitive skills between video game players and non-video game players examined in this study. The only significant difference was found in general attention and the ability to identify a briefly shown target. The test battery used in this study can be used as a basis for more research in this field, which is clearly needed in order to get more clarity in the still ongoing debate about video games and their possible used beyond pure entertainment purposes.

References

Barry, G., Galna, B., & Rochester, L. (2014). The role of exergaming in Parkinson's disease rehabilitation: A systematic review of the evidence. *Journal of NeuroEngineering and Rehabilitation, 11,* art. no. 33.

Basak, C., Boot, W. R., Voss, M. W., & Kramer, A. F. (2008). Can training in a real-time strategy video game attenuate cognitive decline in older adults? *Psychology and Aging, 23,* 765-777.

Belchior, P., Marsiske, M., Sisco, S., Yam, A., Balvelier, D., Ball, K., & Mann, W. (2013). Video game training to improve selective visual attention in older adults. *Computers in Human Behavior, 29*, 1318-1324.

Blacker, K. J., Curby, K. M., Klobusicky, E., & Chein, J. M. (2014). Effects of Action Video Game Training on Visual Working Memory. *Journal of Experimental Psychology: Human Perception and Performance, 40*, 1992-2004.

Boot, W., Kramer, A., Simons, D., Fabiani, M., & Gratton, G. (2008). The effects of video game playing on attention, memory, and executive control. *Acta Psychologica, 129*, 387-398.

Castel, A. D., Pratt, J., & Drummond, E. (2005). The effects of action video game experience on the time course of inhibition of return and the efficiency of visual search. *Acta Psychologica, 119*, 217-230.

Chan, P. A., & Rabinowitz, T. (2006). A cross-sectional analysis of video games and attention deficit hyperactivity disorder symptoms in adolescents. *Annals of General Psychiatry, 5*, 16.

EA DICE. (2016). Battlefield 1. Redwood City, California, USA: Electronic Arts.

Ferguson, C. (2007). The Good, The Bad and the Ugly: A Meta-analytic Review of Positive and Negative Effects of Violent Video Games. *Psychiatr Q*, 309-316.

Gamurs Pty Ltd. (2017, 03 02). *CS:GO Ranks Distribution - Last 30 Days.* Retrieved from CS:GO Squad: https://csgosquad.com/ranks

Green, C. S., & Balevier, D. (2006). Effect of Action Video Games on the Spatial Distribution of Visuospatial Attention. *Journal of Experimental Psychology: Human Perception and Performance. 37*, 1465-1478.

Green, C. S., & Bavelier, D. (2007). Action-Video-Game Experience Alters the Spatial Resolution of Vision. *Psychological Science, 18*, 88-94.

Green, C., & Bavelier, D. (2003). Action video game modifies visual selective attention. *Nature*(432), 534-537.

Heckman, J. (1990). Varieties of Selection Bias. *The American Economic Review, 80*, 313-318.

Jones, K. T., & Berryhill, M. E. (2012). Parietal contributions to visual working memory depend on task difficulty. *Frontiers in Psychiatry, 3*.

Liu, X., Banich, M. T., Jacobson, B. L., & Tanabe, J. L. (2004). Common and distinct neural substrates of attentional control in an integrated Simon and spatial Stroop task as assessed by event-related fMRI. *NeuroImage*, 1097-1106.

Mall, H. (2015). Development methods: Computer-based corpsman training system. In T. S. Hussain, & S. L. Coleman, *Design and Development of Training Games* (pp. 419-427). Cambridge: Cambridge University Press.

Marshall, S. J., Biddle, S. J., Gorely, T., Cameron, N., & Murdey, I. (2004). Relationships between media use, body fatness and physical activity in children and youth: a meta-analysis. *International Journal of Obesity, 28*, 1238-1246.

Ogawa, E. F., You, T., & Leveille, S. G. (2016). Potential benefits of exergaming for cognition and dual-task function in older adults: A systematic review. *Journal of Aging and Physical Activity, 24,* 332-336.

Powers, K. L., Brooks, P. J., Aldrich, N. J., Palladino, M. A., & Alfieri, L. (2013). Effects of video-game play on information processing: A meta-analytic investigation. *Psychonomic Bulletin & Review, 20*, 1055-1079.

Raymond, J. E., Shapiro, K. L., & Arnell, K. M. (1992). Temporary Suppression of Visual Processing in an RSVP Task: An Attentional Blink? *Journal of Experimental Psychology, 18*, 849-800.

Shao, H., Xu, S., Zhang, J., Zheng, J. , Huang, Y., Chen,, J., . . . Ying, Q. (2015). Association between duration of playing video games and bone mineral density in chinese adolescents. *Journal of Clinical Densitometry, 18*, 198-202.

Steamspy. (2017, March 02). *Netherlands*. Retrieved from Steamspy: https://steamspy.com/country/NL

Steiner, N. J., Sheldrick, R. C., Gotthelf, D., & Perrin, E. C. (2011). Computer-based attention training in the schools for children with attention deficit/hyperactivity disorder: A preliminary trial. *Clinical Pedriatics, 50*, 615-622.

Strobach, T., Frensch, P. A., & Schubert, T. (2012). Video game practice optimizes executive control skills in dual-task and task switching situations. *Acta Psychologica, 140*, 13-24.

Valve. (2016, 08 13). *The International*. Retrieved from DOTA 2: http://www.dota2.com/international/overview/

Valve; Hidden Path Entertainment. (2012). Counter-Strike: Global Offensive. Bellevue, Washington, USA: Valve.

Van Hattum, N. (2017, 06 13). *Kijken: Van gamen word je slimmer.* Retrieved from nrc.nl: https://www.nrc.nl/nieuws/2017/06/13/kijken-van-gamen-word-je-slimmer-a1562801

Van Ravenzwaaij, D., Boekel, W., Forstmann, B., Ratcliff, R., & Wagenmakers, E.-J. (2014). Action Video Games Do Not Improve the Speed of Information Processing in Simple Perceptual Tasks. *Journal of Experimental Psychology: General, 143*, 1794-1805.

Appendix A

Script for the directional Stroop task

```
import pygame
import sys
from time import time, clock
import random
from pygame.locals import *
from pygame.compat import unichr_, unicode_
import os



print (os.path.isfile("./" + str(1) + ".txt"))


##### VARIABLES #####
# Colors
col_white = (250, 250, 250)
col_black = (0, 0, 0)
col_gray = (220, 220, 220)
col_red = (255, 0, 0)
col_green = (0, 255, 0)
col_blue = (0, 0, 250)
col_yellow = (250,250,0)


POSITION    = ("down", "up")

COLORS   = {"red": col_red,
            "green": col_green,
            "blue": col_blue}

KEYS     = {"up": K_UP,
            "down": K_DOWN}


BACKGR_COL = col_white
SCREEN_WIDTH = 1920
SCREEN_HEIGHT = 1080
SCREEN_SIZE = (SCREEN_WIDTH , SCREEN_HEIGHT)

pygame.init()
pygame.display.set_mode(SCREEN_SIZE, pygame.FULLSCREEN)
pygame.display.set_caption("Spatial Stroop Task")

screen = pygame.display.get_surface()
screen.fill(BACKGR_COL)
```

```
font = pygame.font.Font(None, 80)
font_small = pygame.font.Font(None, 40)


def main():

    STATE = "welcome"
    trial_number = 0
    reaction_times_list = []

    while True:
        pygame.display.get_surface().fill(BACKGR_COL)

        for event in pygame.event.get():
        # wait for space to be pressed, goes to presenting
            if STATE == "welcome":
                if event.type == KEYDOWN and event.key == K_SPACE:
                    STATE = "prepare_next_trial"
                    print(STATE)

            if STATE == "prepare_next_trial":
                # words are pesented, goes to waiting for response
                trial_number = trial_number + 1
                this_pos  = pick_position()
                this_dir = pick_position()
                time_when_presented = time()
                time_when_stim_presented= clock()
                print (time_when_stim_presented)
                STATE = "wait_for_response"
                print(STATE)

            if STATE == "wait_for_response":
                time_wait= time()
                # decides wether input is correct, goes to feedback
                if event.type == KEYDOWN and event.key in
KEYS.values():
                    time_when_reacted = time()
                    this_reaction_time = time_when_reacted -
time_when_presented
                    this_correctness = (event.key == KEYS[this_pos])
                    STATE = "feedback"
                    print(STATE)

            if STATE == "feedback":
                # decides wether to continue or stop with exercise
                if event.type == KEYDOWN and event.key == K_SPACE:
                    reaction_times_list.append(trial_number)
                    reaction_times_list.append(this_pos == this_dir)
```

```
                    reaction_times_list.append(this_reaction_time)
                    reaction_times_list.append(this_correctness)
                    if trial_number < 5:
                        STATE = "prepare_next_trial"
                    else:
                        STATE = "quit"
                    print(STATE)

            if event.type == QUIT:
                STATE = "quit"

        if STATE == "wait_for_response" and time_wait < (time()-2):
            time_when_reacted = time()
            this_reaction_time = time_when_reacted -
time_when_presented
            this_correctness = False
            STATE= "feedback"
            print(STATE)

        if STATE== "feedback" and  time_wait < (time()-4):
            reaction_times_list.append(trial_number)
            reaction_times_list.append(this_pos == this_dir)
            reaction_times_list.append(this_reaction_time)
            reaction_times_list.append(this_correctness)
            if trial_number < 5:
                 STATE = "prepare_next_trial"
            else:
                 STATE = "quit"
            print(STATE)

        if STATE == "prepare_next_trial":
                # words are pesented, goes to waiting for response
                trial_number = trial_number + 1
                this_pos  = pick_position()
                this_dir = pick_position()
                time_when_presented = time()
                time_when_stim_presented= clock()
                print (time_when_stim_presented)
                STATE = "wait_for_response"
                print(STATE)
                time_wait = time()

        # Drawing to the screen
        if STATE == "welcome":
            draw_welcome()
            draw_button(SCREEN_SIZE[0]*1/3, 700, "UP: Arrow up",
col_red)
            draw_button(SCREEN_SIZE[0]*2/3, 700, "DOWN: Arrow down",
col_red)
```

```
        if STATE == "wait_for_response":
            draw_stimulus(this_pos, this_dir)
        if STATE == "feedback":
            draw_feedback(this_correctness, this_reaction_time)

        if STATE == "goodbye":
            draw_goodbye()

        if STATE == "quit":
            save_into_file(reaction_times_list)
            #pygame.quit()
            #sys.exit()
            import AttentionalBlink.py
            AttentionalBlink.main()

        pygame.display.update()

def pick_position():
    random_number = random.randint(0,1)
    return POSITION[random_number]

def pick_direction():
    random_number = random.randint(0,1)
    return POSITION[random_number]

def draw_button(xpos, ypos, label, color):
    text = font_small.render(label, True, color, col_white)
    text_rectangle = text.get_rect()
    text_rectangle.center = (xpos, ypos)
    screen.blit(text, text_rectangle)

def draw_welcome():
    text_surface = font.render("Spatial STROOP Experiment", True,
col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
    screen.blit(text_surface, text_rectangle)
    text_surface = font.render("Indicate whether the arrow is in the
bottom or upper half.", True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,400)
    screen.blit(text_surface, text_rectangle)
    text_surface = font.render("Disregard the arrow's direction.",
True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,450)
    screen.blit(text_surface, text_rectangle)
```

```
    text_surface = font_small.render("Press Spacebar to continue",
True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,600)
    screen.blit(text_surface, text_rectangle)

def draw_stimulus(pos, dir):
    if pos == "down" and dir == "up":
        #pos down, dir up
        #700, 500
        #pygame.draw.polygon(screen, (0, 0, 0), ((280, 350), (315,
350),(315, 450), (385, 450), (385, 350), //(420, 350), (350, 250)))
        pygame.draw.polygon(screen, (0,0,0), ((SCREEN_WIDTH*0.4,
SCREEN_HEIGHT*0.7), (SCREEN_WIDTH*0.45, SCREEN_HEIGHT*0.7),
(SCREEN_WIDTH*0.45, SCREEN_HEIGHT*0.9), (SCREEN_WIDTH*0.55,
SCREEN_HEIGHT*0.9), (SCREEN_WIDTH*0.55, SCREEN_HEIGHT*0.7),
(SCREEN_WIDTH*0.6, SCREEN_HEIGHT*0.7), (SCREEN_WIDTH*0.5,
SCREEN_HEIGHT*0.5)))
    elif pos == "down" and dir == "down":
        #pos down, dir down
        #pygame.draw.polygon(screen, (0, 0, 0), ((280, 350), (350,
450),(420, 350), (385, 350), (385, 250), (315, 250), (315, 350)))
        pygame.draw.polygon(screen, (0, 0, 0), ((SCREEN_WIDTH*0.4,
SCREEN_HEIGHT*0.7), (SCREEN_WIDTH*0.5,
SCREEN_HEIGHT*0.9),(SCREEN_WIDTH*0.6, SCREEN_HEIGHT*0.7),
(SCREEN_WIDTH*0.55, SCREEN_HEIGHT*0.7), (SCREEN_WIDTH*0.55,
SCREEN_HEIGHT*0.5), (SCREEN_WIDTH*0.45, SCREEN_HEIGHT*0.5),
(SCREEN_WIDTH*0.45, SCREEN_HEIGHT*0.7)))
    elif pos == "up" and dir == "down":
        #pos up, dir down
        #pygame.draw.polygon(screen, (0, 0, 0), ((280, 150), (350,
250),(420, 150), (385, 150), (385, 50), (315, 50), (315, 150)))
        pygame.draw.polygon(screen, (0, 0, 0), ((SCREEN_WIDTH*0.4,
SCREEN_HEIGHT*0.3), (SCREEN_WIDTH*0.5,
SCREEN_HEIGHT*0.5),(SCREEN_WIDTH*0.6, SCREEN_HEIGHT*0.3),
(SCREEN_WIDTH*0.55, SCREEN_HEIGHT*0.3), (SCREEN_WIDTH*0.55,
SCREEN_HEIGHT*0.1), (SCREEN_WIDTH*0.45, SCREEN_HEIGHT*0.1),
(SCREEN_WIDTH*0.45, SCREEN_HEIGHT*0.3)))
    elif pos == "up" and dir == "up":
        #pos up, dir up
        #pygame.draw.polygon(screen, (0, 0, 0), ((280, 150), (315,
150),(315, 250), (385, 250), (385, 150), (420, 150), (350, 50)))
        pygame.draw.polygon(screen, (0, 0, 0), ((SCREEN_WIDTH*0.4,
SCREEN_HEIGHT*0.3), (SCREEN_WIDTH*0.45,
SCREEN_HEIGHT*0.3),(SCREEN_WIDTH*0.45, SCREEN_HEIGHT*0.5),
(SCREEN_WIDTH*0.55, SCREEN_HEIGHT*0.5), (SCREEN_WIDTH*0.55,
SCREEN_HEIGHT*0.3), (SCREEN_WIDTH*0.6, SCREEN_HEIGHT*0.3),
(SCREEN_WIDTH*0.5, SCREEN_HEIGHT*0.1)))
```

```
def draw_feedback(correct, reaction_time):
    if correct:
        text_surface = font_small.render("correct", True, col_black,
BACKGR_COL)
        text_rectangle = text_surface.get_rect()
        text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
        screen.blit(text_surface, text_rectangle)
        text_surface = font_small.render(str(int(reaction_time *
1000)) + "ms", True, col_black, BACKGR_COL)
        text_rectangle = text_surface.get_rect()
        text_rectangle.center = (SCREEN_SIZE[0]/2.0,200)
        screen.blit(text_surface, text_rectangle)
    else:
        text_surface = font_small.render("Wrong key!", True,
col_red, BACKGR_COL)
        text_rectangle = text_surface.get_rect()
        text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
        screen.blit(text_surface, text_rectangle)
    text_surface = font_small.render("Press Spacebar to continue",
True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,300)
    screen.blit(text_surface, text_rectangle)

def draw_goodbye():
    text_surface = font_small.render("END OF THE EXPERIMENT", True,
col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
    screen.blit(text_surface, text_rectangle)
    text_surface = font_small.render("Close the application.", True,
col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,200)
    screen.blit(text_surface, text_rectangle)



def save_into_file(times):
    i = 1
    while os.path.isfile("./Participant" + str(i) + ".txt") == True:
        i = i + 1

    openFunction = open("Participant" + str(i) + ".txt", "w")
    openFunction.write(str("Stroop_trial_number, " +
"Stroop_Congruence, " + "Stroop_This_reaction_time, " +
"Stroop_this_correctness" + "\n"))
    openFunction.write(str(",".join(map(lambda x: str(x), times))))
    openFunction.close()
main()
```

Script for the attentional blink

```
import pygame
import sys
import numpy
from time import time, clock
import time as tm
import random
from pygame.locals import *
from pygame.compat import unichr_, unicode_
import os



#trial kein speichern
#instructions - back and forth
#only look ant x when letter recognized
#doesnt have to be a red x, can jjust be a second black letter etc.
#shapiro raymond

col_white = (255, 255, 255)
col_black = (0, 0, 0)
col_gray = (220, 220, 220)
col_red = (255, 0, 0)
col_green = (0, 255, 0)
col_blue = (0, 0, 250)
col_yellow = (250,250,0)


KEYS     = {"No": K_n,
            "Yes": K_z}

#might need to change layout to nl/en one


BACKGR_COL = col_gray

SCREEN_HEIGHT = 720
SCREEN_WIDTH = int(SCREEN_HEIGHT * 1.777)
SCREEN_SIZE = (SCREEN_WIDTH, SCREEN_HEIGHT)

pygame.init()
pygame.display.set_mode(SCREEN_SIZE, pygame.FULLSCREEN)
pygame.display.set_caption("Attentional Blink")

screen = pygame.display.get_surface()
screen.fill(BACKGR_COL)
```

```
font = pygame.font.Font(None, 80)
font_small = pygame.font.Font(None, 40)

def main():

    STATE = "welcome"
    trial_number = 0
    reaction_times_list = []
    all_letters =
["a","b","c","d","e","f","g","h","i","j","k","l","m","o","p","q","r"
,"s","t","v","x"]
    #initialize values for test with special values first, then if
statement to assign before presentation once, other than that do it
in feedback
    x_bool =  997
    letters = []
    letter = 997
    letter_rnd = 997
    x_pos = 997

    while True:
        pygame.display.get_surface().fill(BACKGR_COL)
        # Changing states
        for event in pygame.event.get():
            if STATE == "welcome":
                 if event.type == KEYDOWN and event.key == K_SPACE:
                     STATE = "prepare_next_trial"
                     print(STATE)
                     pygame.event.clear(pygame.KEYDOWN)
                     print (pygame.event.peek())

            if STATE == "prepare_next_trial":
               print ("start of prepare_next_trial")
               # words are pesented, goes to waiting for response
               if x_bool == 997:
                   x_bool =  random.randint(0,1)
                   print ("x_bool first assigned: " + str(x_bool))
               if letters == []:
                   letters = numpy.random.choice(all_letters, 30)
               if letter_rnd == 997:
                   letter_rnd = random.randint(0,21)
               if letter == 997:
                   letter = letters[letter_rnd]
               if x_pos == 997:
                   x_pos = random.randint(1,8)
               #ugly workaround for script skipping first stimuli
draw
               if trial_number == 0:
```

```
                   draw_stimulus(letters, letter, letter_rnd,
x_bool, x_pos)
               print ("letters: " + str(letters))
               print ("letter: " + str(letter))
               trial_number = trial_number + 1
               time_when_presented = time()
               #time_when_stim_presented = clock()
               #print (time_when_stim_presented)
               STATE = "wait_for_response"
               print(STATE)

            if STATE == "wait_for_response":
                if event.type == KEYDOWN and
(pygame.key.name(event.key)) == letter:
                    print ("correct")
                    this_correctness = True
                    STATE = "wait_for_response_for_X"
                    tm.sleep(0.1)

                elif event.type == KEYDOWN and
(pygame.key.name(event.key)) != letter and event.key != K_SPACE:
                    print ("nope")
                    this_correctness = False
                    STATE =  "wait_for_response_for_X"
                    tm.sleep(0.1)

            if STATE == "wait_for_response_for_X":
               #currently carries over input form wait for respnse,
resulting in one input beeing used for two inquiries
               print (STATE)
               #time_wait = time()
               # decides wether input is correct, goes to feedback
               if event.type == KEYDOWN and ((event.key ==
KEYS["Yes"] and x_bool == 1) or (event.key == KEYS["No"] and x_bool
== 0)):
                   this_correctness_X = True
                   print (this_correctness_X)
                   STATE = "feedback"
                   print(STATE)
               elif event.type == KEYDOWN and ((event.key ==
KEYS["Yes"] and x_bool == 0) or (event.key == KEYS["No"] and x_bool
== 1)):
                   this_correctness_X = False
                   print (this_correctness_X)
                   STATE = "feedback"
                   print (STATE)

            if STATE == "feedback":
```

```
               # decides wether to continue or stop with exercise
               #this_correctness = (event.key == KEYS[this_color])
               if event.type == KEYDOWN and event.key == K_SPACE:
#KEYS[1] and KEYS[1] == x_bool:    #is answer to x stimulus valid?
                   #saving into the file
                   reaction_times_list.append(trial_number)
                   reaction_times_list.append(this_correctness)
                   reaction_times_list.append(x_bool)
                   reaction_times_list.append(x_pos)
                   reaction_times_list.append(this_correctness_X)
                   x_bool =  random.randint(0,1)
                   letter_rnd = random.randint(0,21)
                   letters = numpy.random.choice(all_letters, 30)
                   letter = letters[letter_rnd]
                   x_pos = random.randint(1,8)
                   print ("x_bool re-assigned in feedback: " +
str(x_bool))
                   if trial_number < 5:
                        STATE = "prepare_next_trial"
                   else:
                        STATE = "quit"
                   print(STATE)

            if event.type == QUIT:
                STATE = "quit"
# Drawing to the screen

        if STATE == "welcome":
            draw_welcome()
            draw_button(SCREEN_SIZE[0]*1/6, 600, "Yes: Y",
col_green)
            draw_button(SCREEN_SIZE[0]*5/6, 600, "No: N", col_red)

        if STATE == "prepare_next_trial":
            print ("draw stim/prepare next")
            draw_stimulus(letters, letter, letter_rnd, x_bool,
x_pos)

        if STATE == "wait_for_response":
            draw_wait()

        if STATE == "wait_for_response_for_X":
            draw_wait_for_X()

        if STATE == "feedback":
            draw_feedback(this_correctness, this_correctness_X)

        if STATE == "goodbye":
            draw_goodbye()
```

```
        if STATE == "quit":
            save_into_file(reaction_times_list)
            import DelayedRecognition.py
            DelayedRecognition.main()
            #pygame.quit()
            #sys.exit()

        pygame.display.update()

def draw_button(xpos, ypos, label, color):
    text = font_small.render(label, True, color, BACKGR_COL)
    text_rectangle = text.get_rect()
    text_rectangle.center = (xpos, ypos)
    screen.blit(text, text_rectangle)

def draw_welcome():
    text_surface = font.render("Attentional Blink", True, col_black,
BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
    screen.blit(text_surface, text_rectangle)
    text_surface = font.render("Indicate which letter was white.",
True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,400)
    screen.blit(text_surface, text_rectangle)
    text_surface = font.render("After, indicate whether or not a
number", True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,470)
    screen.blit(text_surface, text_rectangle)
    text_surface = font.render(" was part of the sequence.", True,
col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,540)
    screen.blit(text_surface, text_rectangle)
    text_surface = font_small.render("Press Spacebar to continue",
True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,700)
    screen.blit(text_surface, text_rectangle)

def draw_stimulus(letter_list, letter, letter_pos, x_bool, x_pos):
    for i in range(0, 30):
        #print ("letter_pos: " + str(letter_pos))
        #print ("x_pos: " + str(x_pos))
        #print ("i: " + str(i))
```

```
        if i == letter_pos:
            #print ("looped_black")
            text_surface = font.render(letter, True, col_white,
BACKGR_COL)
            text_rectangle = text_surface.get_rect()
            text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
            screen.blit(text_surface, text_rectangle)
            pygame.display.update()
            tm.sleep(0.033)
            screen.fill(BACKGR_COL)
            pygame.display.update()
            tm.sleep(0.050)
        elif x_bool == 1 and (x_pos + letter_pos) == i:
            #print ("looped_X")
            text_surface = font.render(str(random.randint(0,9)),
True, col_black, BACKGR_COL)
            text_rectangle = text_surface.get_rect()
            text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
            screen.blit(text_surface, text_rectangle)
            pygame.display.update()
            tm.sleep(0.033)
            screen.fill(BACKGR_COL)
            pygame.display.update()
            tm.sleep(0.050)
        else:
            #print ("looped_no_X")
            text_surface = font.render(letter_list[i], True,
col_black, BACKGR_COL)
            text_rectangle = text_surface.get_rect()
            text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
            screen.blit(text_surface, text_rectangle)
            pygame.display.update()
            tm.sleep(10.033)
            screen.fill(BACKGR_COL)
            pygame.display.update()
            tm.sleep(10.050)

"""  alternative for second stimuli, by presenting a red x instead
        elif x_bool == 1 and (x_pos + letter_pos) == i:
            #print ("looped_X")
            text_surface = font.render("X", True, col_red,
BACKGR_COL)
            text_rectangle = text_surface.get_rect()
            text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
            screen.blit(text_surface, text_rectangle)
            pygame.display.update()
            tm.sleep(0.1)
            screen.fill(BACKGR_COL)
            pygame.display.update()
```

```
"""
def draw_wait():
    text_surface = font.render("Which letter was white?", True,
col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
    screen.blit(text_surface, text_rectangle)

def draw_wait_for_X():
    text_surface = font.render("Was there a number?", True,
col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
    screen.blit(text_surface, text_rectangle)

def draw_feedback(correct, correct_X):
    if correct and correct_X:
        text_surface = font_small.render("correct", True, col_black,
BACKGR_COL)
        text_rectangle = text_surface.get_rect()
        text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
        screen.blit(text_surface, text_rectangle)
        text_surface = font_small.render("Press Spacebar to
continue", True, col_black, BACKGR_COL)
        text_rectangle = text_surface.get_rect()
        text_rectangle.center = (SCREEN_SIZE[0]/2.0,300)
        screen.blit(text_surface, text_rectangle)
    else:
        text_surface = font_small.render("incorrect", True, col_red,
BACKGR_COL)
        text_rectangle = text_surface.get_rect()
        text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
        screen.blit(text_surface, text_rectangle)
        text_surface = font_small.render("Press Spacebar to
continue", True, col_black, BACKGR_COL)
        text_rectangle = text_surface.get_rect()
        text_rectangle.center = (SCREEN_SIZE[0]/2.0,300)
        screen.blit(text_surface, text_rectangle)

def draw_goodbye():
    text_surface = font_small.render("END OF THIS EXPERIMENT", True,
col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
    screen.blit(text_surface, text_rectangle)
    text_surface = font_small.render("Close the application to
continue.", True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,200)
```

```
    screen.blit(text_surface, text_rectangle)
#--------------------------------------------------------------------
--------------------------------------------------------------------
-------------------------------
def save_into_file(times):
    i = 1
    while os.path.isfile("./Participant" + str(i) + ".txt") == True:
        i = i + 1

    openFunction = open("Participant" + str(i) + ".txt", "w")
    openFunction.write("Blink_trial_number," + "Blink_correct
letter," + "Blink_x_bool," + "Blink_x/int_pos," + "Blink_correct
X/int" + "\n")
    openFunction.write(str(",".join(map(lambda x: str(x), times))))
    openFunction.close()

main()
```

Script for the change detection task

```
import pygame
import sys
from time import time, clock
import random
from pygame.locals import *
from pygame.compat import unichr_, unicode_
import os
import time as tm

#avoid after-image by maask
#useful fov - geht weil abstand

print (os.path.isfile("./" + str(1) + ".txt"))

##### VARIABLES #####
# Colors
col_white = (250, 250, 250)
col_black = (0, 0, 0)
col_gray = (220, 220, 220)
col_red = (255, 0, 0)
col_green = (0, 255, 0)
col_blue = (0, 0, 250)
col_yellow = (250,250,0)

COLORS   = [col_white, col_black, col_gray, col_red, col_green,
col_blue, col_yellow]
```

```
COLORS_STIM = [col_black, col_gray, col_red, col_green, col_blue,
col_yellow]

KEYS     = {"No": K_n,
            "Yes": K_z}


BACKGR_COL = col_white
SCREEN_WIDTH = 700
SCREEN_HEIGHT = 1200
SCREEN_SIZE = (SCREEN_HEIGHT, SCREEN_WIDTH)

pygame.init()
pygame.display.set_mode(SCREEN_SIZE, pygame.FULLSCREEN)
pygame.display.set_caption("Delayed Recognition")

screen = pygame.display.get_surface()
screen.fill(BACKGR_COL)

font = pygame.font.Font(None, 80)
font_small = pygame.font.Font(None, 40)


def main():
    STATE = "welcome"
    trial_number = 0
    reaction_times_list = []
    square_number_list = []
    color_list = []
    square_number_list_len = 5
    square_bool = 997

    while True:
        pygame.display.get_surface().fill(BACKGR_COL)
        # Changing states
        for event in pygame.event.get():
        # wait for space to be pressed, goes to presenting
            if STATE == "welcome":
                if event.type == KEYDOWN and event.key == K_SPACE:
                    STATE = "prepare_next_trial"
                    print(STATE)

            if STATE == "prepare_next_trial":
                # words are pesented, goes to waiting for response

                if square_bool == 997:
                    square_bool = random.randint(0,1)
                    print ("square_bool first assigned: " +
str(square_bool))
```

```
                trial_number = trial_number + 1
                color_list.clear()
                while square_number_list_len > len(color_list):
                    color_to_add = COLORS_STIM[random.randint(0,5)]
                    if color_to_add not in color_list:
                        color_list.append(color_to_add)
                time_when_presented = time()
                time_when_stim_presented= clock()
                square_number_list.clear()
                while square_number_list_len >
len(square_number_list):
                    number_to_add = random.randint(0,24)
                    if number_to_add not in square_number_list:
                        square_number_list.append(number_to_add)
                print (square_number_list)
                print (time_when_stim_presented)
                STATE = "wait_for_response"
                print(STATE)
                time_wait= time()

            if STATE == "actually_wait_for_response":
                # decides wether input is correct, goes to feedback
                if event.type == KEYDOWN and event.key in
KEYS.values():
                    time_when_reacted = time()
                    this_reaction_time = time_when_reacted -
time_when_presented
                    if event.type == KEYDOWN and event.key == K_z:
                        if square_bool == True:
                            this_correctness = True
                        else:
                            this_correctness = False
                        STATE = "feedback"
                        print(STATE)
                    if event.type == KEYDOWN and event.key == K_n:
                        if square_bool == True:
                            this_correctness = False
                        else:
                            this_correctness = True
                        STATE = "feedback"
                        print(STATE)

            if STATE == "feedback":
                print (STATE)

                # decides wether to continue or stop with exercise

                if event.type == KEYDOWN and event.key == K_SPACE:
                    reaction_times_list.append(trial_number)
```

```
                    reaction_times_list.append(square_bool)
                    reaction_times_list.append(this_correctness)
                    square_bool = random.randint(0,1)
                    print ("square_bool re-assigned in feedback: " +
str(square_bool))
                    if trial_number < 5:
                        STATE = "prepare_next_trial"
                        print (STATE)
                    else:
                        STATE = "quit"
                    print(STATE)

            if event.type == QUIT:
                STATE = "quit"

        # Drawing to the screen
        if STATE == "welcome":
            draw_welcome()

        if STATE == "wait_for_response":
            #ugly workaround for squares flashing all over, becuase
of re-assignment of square_number every loop
            print ("square_bool: " + str(square_bool))
            draw_stimulus(color_list, square_number_list)
            pygame.display.update()
            tm.sleep(1)
            draw_mask()
            pygame.display.update()
            tm.sleep(1)
            draw_stimulus_2(color_list, square_number_list,
square_bool)
            pygame.display.update()
            tm.sleep(1)
            draw_mask()
            pygame.display.update()
            tm.sleep(1)
            STATE = "actually_wait_for_response"

        if STATE == "actually_wait_for_response":
            draw_actually_wait_for_response()

        if STATE == "feedback":
            draw_feedback(this_correctness, 1)

        if STATE == "goodbye":
            draw_goodbye()

        if STATE == "quit":
            save_into_file(reaction_times_list)
```

```
            import UsefulFOV.py
            UsefulFOV.main()
            #pygame.quit()
            #sys.exit()

        pygame.display.update()

def pick_color():
    random_number = random.randint(0,2)
    return COLORS[random_number]

def number_picker():
    return random.randint(0,24)

def draw_button(xpos, ypos, label, color):
    text = font_small.render(label, True, color, col_white)
    text_rectangle = text.get_rect()
    text_rectangle.center = (xpos, ypos)
    screen.blit(text, text_rectangle)

def draw_welcome():
    text_surface = font.render("Delayed Recognition", True,
col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
    screen.blit(text_surface, text_rectangle)
    text_surface = font.render("Indicate whether or not the square",
True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,400)
    screen.blit(text_surface, text_rectangle)
    text_surface = font.render("in the second sequence, appeared",
True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,450)
    screen.blit(text_surface, text_rectangle)
    text_surface = font.render("in the first sequence as well.",
True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,500)
    screen.blit(text_surface, text_rectangle)
    text_surface = font_small.render("Press Spacebar to continue",
True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,650)
    screen.blit(text_surface, text_rectangle)

def draw_stimulus(color_list, square_number_list):
    for i in range(0, len(square_number_list)):
```

```
        square_number = square_number_list[i]
        color = color_list[i]
        #first coloumn
        #------------------------------------------------------------
--------------------------------------------------------------------
-------------------------------------------------------------
        if square_number == 0:
            pygame.draw.polygon(screen, color, ((0,0),
(SCREEN_HEIGHT/5, 0), (SCREEN_HEIGHT/5, SCREEN_WIDTH/5 ),(0,
SCREEN_WIDTH/5) ))
        elif square_number == 1:
            pygame.draw.polygon(screen, color, ((0,SCREEN_WIDTH/5),
(SCREEN_HEIGHT/5, SCREEN_WIDTH/5), (SCREEN_HEIGHT/5,
SCREEN_WIDTH*2/5 ),(0, SCREEN_WIDTH*2/5) ))
        elif square_number == 2:
            pygame.draw.polygon(screen, color,
((0,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT/5, SCREEN_WIDTH*2/5),
(SCREEN_HEIGHT/5, SCREEN_WIDTH*3/5 ),(0, SCREEN_WIDTH*3/5) ))
        elif square_number == 3:
            pygame.draw.polygon(screen, color,
((0,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT/5, SCREEN_WIDTH*3/5),
(SCREEN_HEIGHT/5, SCREEN_WIDTH*4/5 ),(0, SCREEN_WIDTH*4/5) ))
        elif square_number == 4:
            pygame.draw.polygon(screen, color,
((0,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT/5, SCREEN_WIDTH*4/5),
(SCREEN_HEIGHT/5, SCREEN_WIDTH ),(0, SCREEN_WIDTH) ))
        #second coloumn
        #------------------------------------------------------------
--------------------------------------------------------------------
-------------------------------------------------------------
        elif square_number == 5:
            pygame.draw.polygon(screen, color, ((SCREEN_HEIGHT/5,0),
(SCREEN_HEIGHT*2/5, 0), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH/5
),(SCREEN_HEIGHT/5, SCREEN_WIDTH/5) ))
        elif square_number == 6:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT/5,SCREEN_WIDTH/5), (SCREEN_HEIGHT*2/5,
SCREEN_WIDTH/5), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH*2/5
),(SCREEN_HEIGHT/5, SCREEN_WIDTH*2/5) ))
        elif square_number == 7:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT/5,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*2/5,
SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH*3/5
),(SCREEN_HEIGHT/5, SCREEN_WIDTH*3/5) ))
        elif square_number == 8:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT/5,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*2/5,
SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH*4/5
),(SCREEN_HEIGHT/5, SCREEN_WIDTH*4/5) ))
```

```
        elif square_number == 9:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT/5,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*2/5,
SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH
),(SCREEN_HEIGHT/5, SCREEN_WIDTH) ))
        #third coloumn
        #------------------------------------------------------------
--------------------------------------------------------------------
------------------------------------------------------------
        elif square_number == 10:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*2/5,0), (SCREEN_HEIGHT*3/5, 0), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH/5 ),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH/5) ))
        elif square_number == 11:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*2/5,SCREEN_WIDTH/5), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH/5), (SCREEN_HEIGHT*3/5, SCREEN_WIDTH*2/5
),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH*2/5) ))
        elif square_number == 12:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*2/5,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*3/5, SCREEN_WIDTH*3/5
),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH*3/5) ))
        elif square_number == 13:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*2/5,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*3/5, SCREEN_WIDTH*4/5
),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH*4/5) ))
        elif square_number == 14:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*2/5,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*3/5, SCREEN_WIDTH
),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH) ))
        #fourth coloumn
        #------------------------------------------------------------
--------------------------------------------------------------------
------------------------------------------------------------
        elif square_number == 15:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*3/5,0), (SCREEN_HEIGHT*4/5, 0), (SCREEN_HEIGHT*4/5,
SCREEN_WIDTH/5 ),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH/5) ))
        elif square_number == 16:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*3/5,SCREEN_WIDTH/5), (SCREEN_HEIGHT*4/5,
SCREEN_WIDTH/5), (SCREEN_HEIGHT*4/5, SCREEN_WIDTH*2/5
),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH*2/5) ))
        elif square_number == 17:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*3/5,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*4/5,
```

```
SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*4/5, SCREEN_WIDTH*3/5
),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH*3/5) ))
        elif square_number == 18:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*3/5,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*4/5,
SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*4/5, SCREEN_WIDTH*4/5
),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH*4/5) ))
        elif square_number == 19:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*3/5,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*4/5,
SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*4/5, SCREEN_WIDTH
),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH) ))
        #fifth coloumn
        #-------------------------------------------------------------
--------------------------------------------------------------------
------------------------------------------------------------v
        elif square_number == 20:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*4/5,0), (SCREEN_HEIGHT, 0), (SCREEN_HEIGHT,
SCREEN_WIDTH/5 ),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH/5) ))
        elif square_number == 21:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*4/5,SCREEN_WIDTH/5), (SCREEN_HEIGHT,
SCREEN_WIDTH/5), (SCREEN_HEIGHT, SCREEN_WIDTH*2/5
),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH*2/5) ))
        elif square_number == 22:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*4/5,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT,
SCREEN_WIDTH*2/5), (SCREEN_HEIGHT, SCREEN_WIDTH*3/5
),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH*3/5) ))
        elif square_number == 23:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*4/5,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT,
SCREEN_WIDTH*3/5), (SCREEN_HEIGHT, SCREEN_WIDTH*4/5
),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH*4/5) ))
        elif square_number == 24:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*4/5,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT,
SCREEN_WIDTH*4/5), (SCREEN_HEIGHT, SCREEN_WIDTH
),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH) ))

def draw_mask():
    screen.fill(col_white)
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((0,0), (SCREEN_HEIGHT/5, 0), (SCREEN_HEIGHT/5, SCREEN_WIDTH/5 ),(0,
SCREEN_WIDTH/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((0,SCREEN_WIDTH/5), (SCREEN_HEIGHT/5, SCREEN_WIDTH/5),
(SCREEN_HEIGHT/5, SCREEN_WIDTH*2/5 ),(0, SCREEN_WIDTH*2/5) ))
```

```
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((0,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT/5, SCREEN_WIDTH*2/5),
(SCREEN_HEIGHT/5, SCREEN_WIDTH*3/5 ),(0, SCREEN_WIDTH*3/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((0,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT/5, SCREEN_WIDTH*3/5),
(SCREEN_HEIGHT/5, SCREEN_WIDTH*4/5 ),(0, SCREEN_WIDTH*4/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((0,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT/5, SCREEN_WIDTH*4/5),
(SCREEN_HEIGHT/5, SCREEN_WIDTH ),(0, SCREEN_WIDTH) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT/5,0), (SCREEN_HEIGHT*2/5, 0), (SCREEN_HEIGHT*2/5,
SCREEN_WIDTH/5 ),(SCREEN_HEIGHT/5, SCREEN_WIDTH/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT/5,SCREEN_WIDTH/5), (SCREEN_HEIGHT*2/5,
SCREEN_WIDTH/5), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH*2/5
),(SCREEN_HEIGHT/5, SCREEN_WIDTH*2/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT/5,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*2/5,
SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH*3/5
),(SCREEN_HEIGHT/5, SCREEN_WIDTH*3/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT/5,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*2/5,
SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH*4/5
),(SCREEN_HEIGHT/5, SCREEN_WIDTH*4/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT/5,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*2/5,
SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH
),(SCREEN_HEIGHT/5, SCREEN_WIDTH) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*2/5,0), (SCREEN_HEIGHT*3/5, 0), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH/5 ),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*2/5,SCREEN_WIDTH/5), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH/5), (SCREEN_HEIGHT*3/5, SCREEN_WIDTH*2/5
),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH*2/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*2/5,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*3/5, SCREEN_WIDTH*3/5
),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH*3/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*2/5,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*3/5, SCREEN_WIDTH*4/5
),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH*4/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*2/5,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*3/5, SCREEN_WIDTH
),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH) ))
```

```
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*3/5,0), (SCREEN_HEIGHT*4/5, 0), (SCREEN_HEIGHT*4/5,
SCREEN_WIDTH/5 ),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*3/5,SCREEN_WIDTH/5), (SCREEN_HEIGHT*4/5,
SCREEN_WIDTH/5), (SCREEN_HEIGHT*4/5, SCREEN_WIDTH*2/5
),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH*2/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*3/5,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*4/5,
SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*4/5, SCREEN_WIDTH*3/5
),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH*3/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*3/5,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*4/5,
SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*4/5, SCREEN_WIDTH*4/5
),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH*4/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*3/5,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*4/5,
SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*4/5, SCREEN_WIDTH
),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*4/5,0), (SCREEN_HEIGHT, 0), (SCREEN_HEIGHT,
SCREEN_WIDTH/5 ),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*4/5,SCREEN_WIDTH/5), (SCREEN_HEIGHT,
SCREEN_WIDTH/5), (SCREEN_HEIGHT, SCREEN_WIDTH*2/5
),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH*2/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*4/5,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT,
SCREEN_WIDTH*2/5), (SCREEN_HEIGHT, SCREEN_WIDTH*3/5
),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH*3/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*4/5,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT,
SCREEN_WIDTH*3/5), (SCREEN_HEIGHT, SCREEN_WIDTH*4/5
),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH*4/5) ))
    pygame.draw.polygon(screen, COLORS_STIM[random.randint(0,4)],
((SCREEN_HEIGHT*4/5,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT,
SCREEN_WIDTH*4/5), (SCREEN_HEIGHT, SCREEN_WIDTH
),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH) ))

def draw_stimulus_2(color_list, square_number_list, square_bool):
        print (square_bool)
        print (color_list[0])
        print (square_number_list[0])
        if square_bool == 1:
            color = color_list[0]
            square_number = square_number_list[0]
        else:
            color = color_list[random.randint(0,4)]
            square_number = random.randint(0,24)
```

```
            while square_number == square_number_list[0]:
                square_number = random.randint(0,24)
        screen.fill(col_white)

        if square_number == 0:
            pygame.draw.polygon(screen, color, ((0,0),
(SCREEN_HEIGHT/5, 0), (SCREEN_HEIGHT/5, SCREEN_WIDTH/5 ),(0,
SCREEN_WIDTH/5) ))
        elif square_number == 1:
            pygame.draw.polygon(screen, color, ((0,SCREEN_WIDTH/5),
(SCREEN_HEIGHT/5, SCREEN_WIDTH/5), (SCREEN_HEIGHT/5,
SCREEN_WIDTH*2/5 ),(0, SCREEN_WIDTH*2/5) ))
        elif square_number == 2:
            pygame.draw.polygon(screen, color,
((0,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT/5, SCREEN_WIDTH*2/5),
(SCREEN_HEIGHT/5, SCREEN_WIDTH*3/5 ),(0, SCREEN_WIDTH*3/5) ))
        elif square_number == 3:
            pygame.draw.polygon(screen, color,
((0,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT/5, SCREEN_WIDTH*3/5),
(SCREEN_HEIGHT/5, SCREEN_WIDTH*4/5 ),(0, SCREEN_WIDTH*4/5) ))
        elif square_number == 4:
            pygame.draw.polygon(screen, color,
((0,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT/5, SCREEN_WIDTH*4/5),
(SCREEN_HEIGHT/5, SCREEN_WIDTH ),(0, SCREEN_WIDTH) ))
        #second coloumn
        #------------------------------------------------------------
--------------------------------------------------------------------
-------------------------------------------------------------
        elif square_number == 5:
            pygame.draw.polygon(screen, color, ((SCREEN_HEIGHT/5,0),
(SCREEN_HEIGHT*2/5, 0), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH/5
),(SCREEN_HEIGHT/5, SCREEN_WIDTH/5) ))
        elif square_number == 6:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT/5,SCREEN_WIDTH/5), (SCREEN_HEIGHT*2/5,
SCREEN_WIDTH/5), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH*2/5
),(SCREEN_HEIGHT/5, SCREEN_WIDTH*2/5) ))
        elif square_number == 7:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT/5,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*2/5,
SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH*3/5
),(SCREEN_HEIGHT/5, SCREEN_WIDTH*3/5) ))
        elif square_number == 8:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT/5,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*2/5,
SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH*4/5
),(SCREEN_HEIGHT/5, SCREEN_WIDTH*4/5) ))
        elif square_number == 9:
```

```
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT/5,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*2/5,
SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*2/5, SCREEN_WIDTH
),(SCREEN_HEIGHT/5, SCREEN_WIDTH) ))
        #third coloumn
        #------------------------------------------------------------
--------------------------------------------------------------------
------------------------------------------------------------
        elif square_number == 10:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*2/5,0), (SCREEN_HEIGHT*3/5, 0), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH/5 ),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH/5) ))
        elif square_number == 11:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*2/5,SCREEN_WIDTH/5), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH/5), (SCREEN_HEIGHT*3/5, SCREEN_WIDTH*2/5
),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH*2/5) ))
        elif square_number == 12:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*2/5,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*3/5, SCREEN_WIDTH*3/5
),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH*3/5) ))
        elif square_number == 13:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*2/5,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*3/5, SCREEN_WIDTH*4/5
),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH*4/5) ))
        elif square_number == 14:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*2/5,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*3/5,
SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*3/5, SCREEN_WIDTH
),(SCREEN_HEIGHT*2/5, SCREEN_WIDTH) ))
        #fourth coloumn
        #------------------------------------------------------------
--------------------------------------------------------------------
------------------------------------------------------------
        elif square_number == 15:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*3/5,0), (SCREEN_HEIGHT*4/5, 0), (SCREEN_HEIGHT*4/5,
SCREEN_WIDTH/5 ),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH/5) ))
        elif square_number == 16:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*3/5,SCREEN_WIDTH/5), (SCREEN_HEIGHT*4/5,
SCREEN_WIDTH/5), (SCREEN_HEIGHT*4/5, SCREEN_WIDTH*2/5
),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH*2/5) ))
        elif square_number == 17:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*3/5,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*4/5,
```

```
SCREEN_WIDTH*2/5), (SCREEN_HEIGHT*4/5, SCREEN_WIDTH*3/5
),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH*3/5) ))
        elif square_number == 18:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*3/5,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*4/5,
SCREEN_WIDTH*3/5), (SCREEN_HEIGHT*4/5, SCREEN_WIDTH*4/5
),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH*4/5) ))
        elif square_number == 19:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*3/5,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*4/5,
SCREEN_WIDTH*4/5), (SCREEN_HEIGHT*4/5, SCREEN_WIDTH
),(SCREEN_HEIGHT*3/5, SCREEN_WIDTH) ))
        #fifth coloumn
        #------------------------------------------------------------
--------------------------------------------------------------------
-----------------------------------------------------------v
        elif square_number == 20:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*4/5,0), (SCREEN_HEIGHT, 0), (SCREEN_HEIGHT,
SCREEN_WIDTH/5 ),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH/5) ))
        elif square_number == 21:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*4/5,SCREEN_WIDTH/5), (SCREEN_HEIGHT,
SCREEN_WIDTH/5), (SCREEN_HEIGHT, SCREEN_WIDTH*2/5
),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH*2/5) ))
        elif square_number == 22:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*4/5,SCREEN_WIDTH*2/5), (SCREEN_HEIGHT,
SCREEN_WIDTH*2/5), (SCREEN_HEIGHT, SCREEN_WIDTH*3/5
),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH*3/5) ))
        elif square_number == 23:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*4/5,SCREEN_WIDTH*3/5), (SCREEN_HEIGHT,
SCREEN_WIDTH*3/5), (SCREEN_HEIGHT, SCREEN_WIDTH*4/5
),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH*4/5) ))
        elif square_number == 24:
            pygame.draw.polygon(screen, color,
((SCREEN_HEIGHT*4/5,SCREEN_WIDTH*4/5), (SCREEN_HEIGHT,
SCREEN_WIDTH*4/5), (SCREEN_HEIGHT, SCREEN_WIDTH
),(SCREEN_HEIGHT*4/5, SCREEN_WIDTH) ))

def draw_actually_wait_for_response():
    text_surface = font_small.render("Did one of the squares re-
appear?", True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
    screen.blit(text_surface, text_rectangle)

def draw_feedback(correct, reaction_time):
```

```
    if correct:
        text_surface = font_small.render("correct", True, col_black,
BACKGR_COL)
        text_rectangle = text_surface.get_rect()
        text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
        screen.blit(text_surface, text_rectangle)
    else:
        text_surface = font_small.render("incorrect", True, col_red,
BACKGR_COL)
        text_rectangle = text_surface.get_rect()
        text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
        screen.blit(text_surface, text_rectangle)
    text_surface = font_small.render("Press Spacebar to continue",
True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,300)
    screen.blit(text_surface, text_rectangle)

def draw_goodbye():
    text_surface = font_small.render("END OF THE EXPERIMENT", True,
col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
    screen.blit(text_surface, text_rectangle)
    text_surface = font_small.render("Close the application.", True,
col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,200)
    screen.blit(text_surface, text_rectangle)
#--------------------------------------------------------------------
---------------------------------------------------------------------
------------------------------
def save_into_file(times):
    i = 1
    while os.path.isfile("./Participant" + str(i) + ".txt") == True:
        i = i + 1

    openFunction = open("Participant" + str(i) + ".txt", "w")
    openFunction.write("Recog_Trial_number" + "Recog_square_bool," +
"Recog_this_correctness," + "\n")
    openFunction.write(str(",".join(map(lambda x: str(x), times))))
    openFunction.close()

main()
```

Script for the useful FOV task

```
import pygame
import sys
from time import time, clock
import random
from pygame.locals import *
from pygame.compat import unichr_, unicode_
import os
import time as tm

#https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2884279/figure/F1/
#use mouseover to indicate which circle lit up
#vielleicht nicht komplett zufällig, wegen peripherie

print (os.path.isfile("./" + str(1) + ".txt"))

##### VARIABLES #####
# Colors
col_white = (250, 250, 250)
col_black = (0, 0, 0)
col_gray = (220, 220, 220)
col_red = (255, 0, 0)
col_green = (0, 255, 0)
col_blue = (0, 0, 250)
col_yellow = (250,250,0)

COLORS   = [col_white, col_black, col_gray, col_red, col_green,
col_blue, col_yellow]

COLORS_STIM = [col_black, col_gray, col_red, col_green, col_blue,
col_yellow]

KEYS     = {"No": K_n,
            "Yes": K_z}



BACKGR_COL = col_white
SCREEN_HEIGHT = 720
SCREEN_WIDTH = int(SCREEN_HEIGHT * 1.777)

SCREEN_SIZE = (SCREEN_WIDTH, SCREEN_HEIGHT)

pygame.init()
pygame.display.set_mode(SCREEN_SIZE, pygame.FULLSCREEN)
pygame.display.set_caption("Useful FOV")

screen = pygame.display.get_surface()
screen.fill(BACKGR_COL)
```

```
font = pygame.font.Font(None, 80)
font_small = pygame.font.Font(None, 40)


def main():
    STATE = "welcome"
    trial_number = 0
    reaction_times_list = []
    circle_number = 997

    while True:
        pygame.display.get_surface().fill(BACKGR_COL)
        # Changing states
        for event in pygame.event.get():
        # wait for space to be pressed, goes to presenting
            if STATE == "welcome":
                if event.type == KEYDOWN and event.key == K_SPACE:
                    STATE = "prepare_next_trial"
                    print(STATE)

            if STATE == "prepare_next_trial":
                # words are pesented, goes to waiting for response

                trial_number = trial_number + 1
                if circle_number == 997:
                    circle_number = random.randint(0,24)
                time_when_presented = time()
                time_when_stim_presented= clock()
                print (circle_number)
                print (time_when_stim_presented)
                STATE = "wait_for_response"
                print(STATE)
                time_wait= time()

            if STATE == "actually_wait_for_response":
                print(STATE)
                # decides wether input is correct, goes to feedback
                if event.type == pygame.MOUSEBUTTONDOWN:
                    x,y = event.pos
                    print (x,y)
                    color = col_black
                    #1,6,7,12,13,18,19,24
                    if pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*4/8)),
50).collidepoint(x,y) and circle_number == 0:
                        this_correctness = True
```

```
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*1/8)),
50).collidepoint(x,y) and circle_number == 1:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*2/8)),
50).collidepoint(x,y) and circle_number == 2:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*3/8)),
50).collidepoint(x,y) and circle_number == 3:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*5/8)),
50).collidepoint(x,y) and circle_number == 4:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*6/8)),
50).collidepoint(x,y) and circle_number == 5:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*7/8)),
50).collidepoint(x,y) and circle_number == 6:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*1/8), int(SCREEN_HEIGHT*4/8)),
50).collidepoint(x,y) and circle_number == 7:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*2/8), int(SCREEN_HEIGHT*4/8)),
50).collidepoint(x,y) and circle_number == 8:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*3/8), int(SCREEN_HEIGHT*4/8)),
50).collidepoint(x,y) and circle_number == 9:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*5/8), int(SCREEN_HEIGHT*4/8)),
50).collidepoint(x,y) and circle_number == 10:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*6/8), int(SCREEN_HEIGHT*4/8)),
50).collidepoint(x,y) and circle_number == 11:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*7/8), int(SCREEN_HEIGHT*4/8)),
50).collidepoint(x,y) and circle_number == 12:
                        this_correctness = True
```

```
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*1/8), int(SCREEN_HEIGHT*1/8)),
50).collidepoint(x,y) and circle_number == 13:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*2/8), int(SCREEN_HEIGHT*2/8)),
50).collidepoint(x,y) and circle_number == 14:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*3/8), int(SCREEN_HEIGHT*3/8)),
50).collidepoint(x,y) and circle_number == 15:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*5/8), int(SCREEN_HEIGHT*5/8)),
50).collidepoint(x,y) and circle_number == 16:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*6/8), int(SCREEN_HEIGHT*6/8)),
50).collidepoint(x,y) and circle_number == 17:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*7/8), int(SCREEN_HEIGHT*7/8)),
50).collidepoint(x,y) and circle_number == 18:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*7/8), int(SCREEN_HEIGHT*1/8)),
50).collidepoint(x,y) and circle_number == 19:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*6/8), int(SCREEN_HEIGHT*2/8)),
50).collidepoint(x,y) and circle_number == 20:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*5/8), int(SCREEN_HEIGHT*3/8)),
50).collidepoint(x,y) and circle_number == 21:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*3/8), int(SCREEN_HEIGHT*5/8)),
50).collidepoint(x,y) and circle_number  == 22:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*2/8), int(SCREEN_HEIGHT*6/8)),
50).collidepoint(x,y) and circle_number == 23:
                        this_correctness = True
                    elif pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*1/8), int(SCREEN_HEIGHT*7/8)),
50).collidepoint(x,y) and circle_number == 24:
                        this_correctness = True
                    else:
```

```
                        this_correctness = False
                    time_when_reacted = time()
                    this_reaction_time = time_when_reacted -
time_when_presented
                    STATE = "feedback"
                    print(STATE)

            if STATE == "feedback":
                print (STATE)



                # decides wether to continue or stop with exercise

                if event.type == KEYDOWN and event.key == K_SPACE:
                    reaction_times_list.append(trial_number)
                    reaction_times_list.append(this_correctness)
                    reaction_times_list.append(circle_number)
                    reaction_times_list.append(this_reaction_time)
                    circle_number = random.randint(0,24)
                    print ("data beeing saved: " + str(trial_number)
+ str(this_correctness) + str(circle_number) +
str(this_reaction_time))
                    if trial_number < 5:
                        STATE = "prepare_next_trial"
                        print (STATE)
                    else:
                        STATE = "goodbye"
                    print(STATE)

            if event.type == QUIT:
                STATE = "quit"

        # Drawing to the screen
        if STATE == "welcome":
            draw_welcome()

        if STATE == "wait_for_response":
            draw_stimulus(col_black)
            pygame.display.update()
            tm.sleep(1)
            draw_stimulus_2(circle_number, col_red)
            pygame.display.update()
            tm.sleep(0.2)
            draw_mask()
            pygame.display.update()
            tm.sleep(1)

            STATE = "actually_wait_for_response"
```

```
        if STATE == "actually_wait_for_response":
            draw_actually_wait_for_response(col_black)

        if STATE == "feedback":
            draw_feedback(this_correctness, 1)

        if STATE == "goodbye":
            draw_goodbye()

        if STATE == "quit":
            save_into_file(reaction_times_list)
            pygame.quit()
            sys.exit()

        pygame.display.update()

def pick_color():
    random_number = random.randint(0,2)
    return COLORS[random_number]

def number_picker():
    return random.randint(0,24)

def draw_button(xpos, ypos, label, color):
    text = font_small.render(label, True, color, col_white)
    text_rectangle = text.get_rect()
    text_rectangle.center = (xpos, ypos)
    screen.blit(text, text_rectangle)

def draw_welcome():
    text_surface = font.render("Useful FOV task", True, col_black,
BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
    screen.blit(text_surface, text_rectangle)
    text_surface = font.render("Indicate which of the circles turned
red.", True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,300)
    screen.blit(text_surface, text_rectangle)
    text_surface = font.render("Fixate on the circle in the
center.", True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,400)
    screen.blit(text_surface, text_rectangle)
    text_surface = font_small.render("Press Spacebar to continue",
True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
```

```
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,600)
    screen.blit(text_surface, text_rectangle)

def draw_stimulus(color):
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*4/8)), 33)

    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*1/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*2/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*3/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*5/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*6/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*7/8)), 33)

    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*1/8),
int(SCREEN_HEIGHT*4/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*2/8),
int(SCREEN_HEIGHT*4/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*3/8),
int(SCREEN_HEIGHT*4/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*5/8),
int(SCREEN_HEIGHT*4/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*6/8),
int(SCREEN_HEIGHT*4/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*7/8),
int(SCREEN_HEIGHT*4/8)), 33)

    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*1/8),
int(SCREEN_HEIGHT*1/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*2/8),
int(SCREEN_HEIGHT*2/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*3/8),
int(SCREEN_HEIGHT*3/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*5/8),
int(SCREEN_HEIGHT*5/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*6/8),
int(SCREEN_HEIGHT*6/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*7/8),
int(SCREEN_HEIGHT*7/8)), 33)

    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*7/8),
int(SCREEN_HEIGHT*1/8)), 33)
```

```
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*6/8), 
int(SCREEN_HEIGHT*2/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*5/8), 
int(SCREEN_HEIGHT*3/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*3/8), 
int(SCREEN_HEIGHT*5/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*2/8), 
int(SCREEN_HEIGHT*6/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*1/8), 
int(SCREEN_HEIGHT*7/8)), 33)

def draw_mask():
    i = 0
    screen.fill(col_white)
    while i < 50:
        pygame.draw.circle(screen, col_black, 
(int(SCREEN_WIDTH*((int(random.randint(1,15)))/16)), 
int(SCREEN_HEIGHT*((int(random.randint(1,15)))/16))), 33)
        i = i +1
    pygame.draw.circle(screen, col_red, 
(int(SCREEN_WIDTH*((int(random.randint(1,15)))/16)), 
int(SCREEN_HEIGHT*((int(random.randint(1,15)))/16))), 33)

def draw_stimulus_2(square_number, color):

        if square_number == 0:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*4/8)), 33)
        elif square_number == 1:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*1/8)), 33)
        elif square_number == 2:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*2/8)), 33)
        elif square_number == 3:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*3/8)), 33)
        elif square_number == 4:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*5/8)), 33)
        #second coloumn
        #-------------------------------------------------------------
--------------------------------------------------------------------
-------------------------------------------------------------
        elif square_number == 5:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*6/8)), 33)
        elif square_number == 6:
```

```
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*4/8), int(SCREEN_HEIGHT*7/8)), 33)
        elif square_number == 7:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*1/8), int(SCREEN_HEIGHT*4/8)), 33)
        elif square_number == 8:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*2/8), int(SCREEN_HEIGHT*4/8)), 33)
        elif square_number == 9:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*3/8), int(SCREEN_HEIGHT*4/8)), 33)
        #third coloumn
        #-----------------------------------------------------------
------------------------------------------------------------------
-----------------------------------------------------------
        elif square_number == 10:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*5/8), int(SCREEN_HEIGHT*4/8)), 33)
        elif square_number == 11:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*6/8), int(SCREEN_HEIGHT*4/8)), 33)
        elif square_number == 12:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*7/8), int(SCREEN_HEIGHT*4/8)), 33)
        elif square_number == 13:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*1/8), int(SCREEN_HEIGHT*1/8)), 33)
        elif square_number == 14:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*2/8), int(SCREEN_HEIGHT*2/8)), 33)
        #fourth coloumn
        #-----------------------------------------------------------
------------------------------------------------------------------
-----------------------------------------------------------
        elif square_number == 15:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*3/8), int(SCREEN_HEIGHT*3/8)), 33)
        elif square_number == 16:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*5/8), int(SCREEN_HEIGHT*5/8)), 33)
        elif square_number == 17:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*6/8), int(SCREEN_HEIGHT*6/8)), 33)
        elif square_number == 18:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*7/8), int(SCREEN_HEIGHT*7/8)), 33)
        elif square_number == 19:
            pygame.draw.circle(screen, color, 
(int(SCREEN_WIDTH*7/8), int(SCREEN_HEIGHT*1/8)), 33)
```

```
        #fifth coloumn
        #------------------------------------------------------------
---------------------------------------------------------------------
-----------------------------------------------------------v
        elif square_number == 20:
            pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*6/8), int(SCREEN_HEIGHT*2/8)), 33)
        elif square_number == 21:
            pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*5/8), int(SCREEN_HEIGHT*3/8)), 33)
        elif square_number == 22:
            pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*3/8), int(SCREEN_HEIGHT*5/8)), 33)
        elif square_number == 23:
            pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*2/8), int(SCREEN_HEIGHT*6/8)), 33)
        elif square_number == 24:
            pygame.draw.circle(screen, color,
(int(SCREEN_WIDTH*1/8), int(SCREEN_HEIGHT*7/8)), 33)

def draw_actually_wait_for_response(color):
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*4/8)), 33)

    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*1/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*2/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*3/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*5/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*6/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*4/8),
int(SCREEN_HEIGHT*7/8)), 33)

    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*1/8),
int(SCREEN_HEIGHT*4/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*2/8),
int(SCREEN_HEIGHT*4/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*3/8),
int(SCREEN_HEIGHT*4/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*5/8),
int(SCREEN_HEIGHT*4/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*6/8),
int(SCREEN_HEIGHT*4/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*7/8),
int(SCREEN_HEIGHT*4/8)), 33)
```

```
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*1/8),
int(SCREEN_HEIGHT*1/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*2/8),
int(SCREEN_HEIGHT*2/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*3/8),
int(SCREEN_HEIGHT*3/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*5/8),
int(SCREEN_HEIGHT*5/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*6/8),
int(SCREEN_HEIGHT*6/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*7/8),
int(SCREEN_HEIGHT*7/8)), 33)

    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*7/8),
int(SCREEN_HEIGHT*1/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*6/8),
int(SCREEN_HEIGHT*2/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*5/8),
int(SCREEN_HEIGHT*3/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*3/8),
int(SCREEN_HEIGHT*5/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*2/8),
int(SCREEN_HEIGHT*6/8)), 33)
    pygame.draw.circle(screen, color, (int(SCREEN_WIDTH*1/8),
int(SCREEN_HEIGHT*7/8)), 33)


def draw_feedback(correct, reaction_time):
    if correct:
        text_surface = font_small.render("correct", True, col_black,
BACKGR_COL)
        text_rectangle = text_surface.get_rect()
        text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
        screen.blit(text_surface, text_rectangle)
        text_surface = font_small.render(str(int(reaction_time *
1000)) + "ms", True, col_black, BACKGR_COL)
        text_rectangle = text_surface.get_rect()
        text_rectangle.center = (SCREEN_SIZE[0]/2.0,200)
        screen.blit(text_surface, text_rectangle)
    else:
        text_surface = font_small.render("incorrect", True, col_red,
BACKGR_COL)
        text_rectangle = text_surface.get_rect()
        text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
        screen.blit(text_surface, text_rectangle)
    text_surface = font_small.render("Press Spacebar to continue",
True, col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
```

```
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,300)
    screen.blit(text_surface, text_rectangle)

def draw_goodbye():
    text_surface = font_small.render("END OF THE EXPERIMENT", True,
col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,150)
    screen.blit(text_surface, text_rectangle)
    text_surface = font_small.render("Close the application.", True,
col_black, BACKGR_COL)
    text_rectangle = text_surface.get_rect()
    text_rectangle.center = (SCREEN_SIZE[0]/2.0,200)
    screen.blit(text_surface, text_rectangle)
#-------------------------------------------------------------------
--------------------------------------------------------------------
------------------------------
def save_into_file(times):
    i = 1
    while os.path.isfile("./Participant" + str(i) + ".txt") == True:
        i = i + 1

    openFunction = open("Participant" + str(i) + ".txt", "w")
    openFunction.write("FOV_trial_number," + "FOV_this_correctness,"
+ "FOV_circle_number," + "FOV_reaction_time," + "\n")
    openFunction.write(str(",".join(map(lambda x: str(x), times))))
    openFunction.close()

main()
```

Appendix B

Questionnaire

Gender: ☐ Male ☐ Female

Age: _____

Nationality: ☐ Dutch ☐ German ☐ Other:____________________

I am experienced in (choose one):

☐ Counter Strike

Playtime(hrs):

☐ 0-150 ☐ 151-300 ☐ 301-450 ☐ 451-600 ☐ 601-750 ☐751-1000 ☐ >1000

Highest rank ever achieved:

- ☐ Unranked
- ☐ Silver I
- ☐ Silver II
- ☐ Silver III
- ☐ Silver IV
- ☐ Silver Elite
- ☐ Silver Elite Master
- ☐ Gold Nova I
- ☐ Gold Nova II
- ☐ Gold Nova III
- ☐ Gold Nova Master
- ☐ Master Guardian I
- ☐ Master Guardian II
- ☐ Master Guardian Elite
- ☐ Distinguished Master Guardian
- ☐ Legendary Eagle
- ☐ Legendary Eagle Master
- ☐ Supreme Master First Class
- ☐ Global Elite

☐ Overwatch

Playtime (hrs):

☐ 0-50 ☐ 51-100 ☐ 101-200 ☐ 201-500 ☐ 501-1000 ☐1000+

Highest rank ever achieved:

- ☐ Bronze
- ☐ Silver
- ☐ Gold
- ☐ Platinum
- ☐ Diamond
- ☐ Master
- ☐ Grandmaster
- ☐ Top 500

☐ I have no experience in action video games

☐ Colour-blind

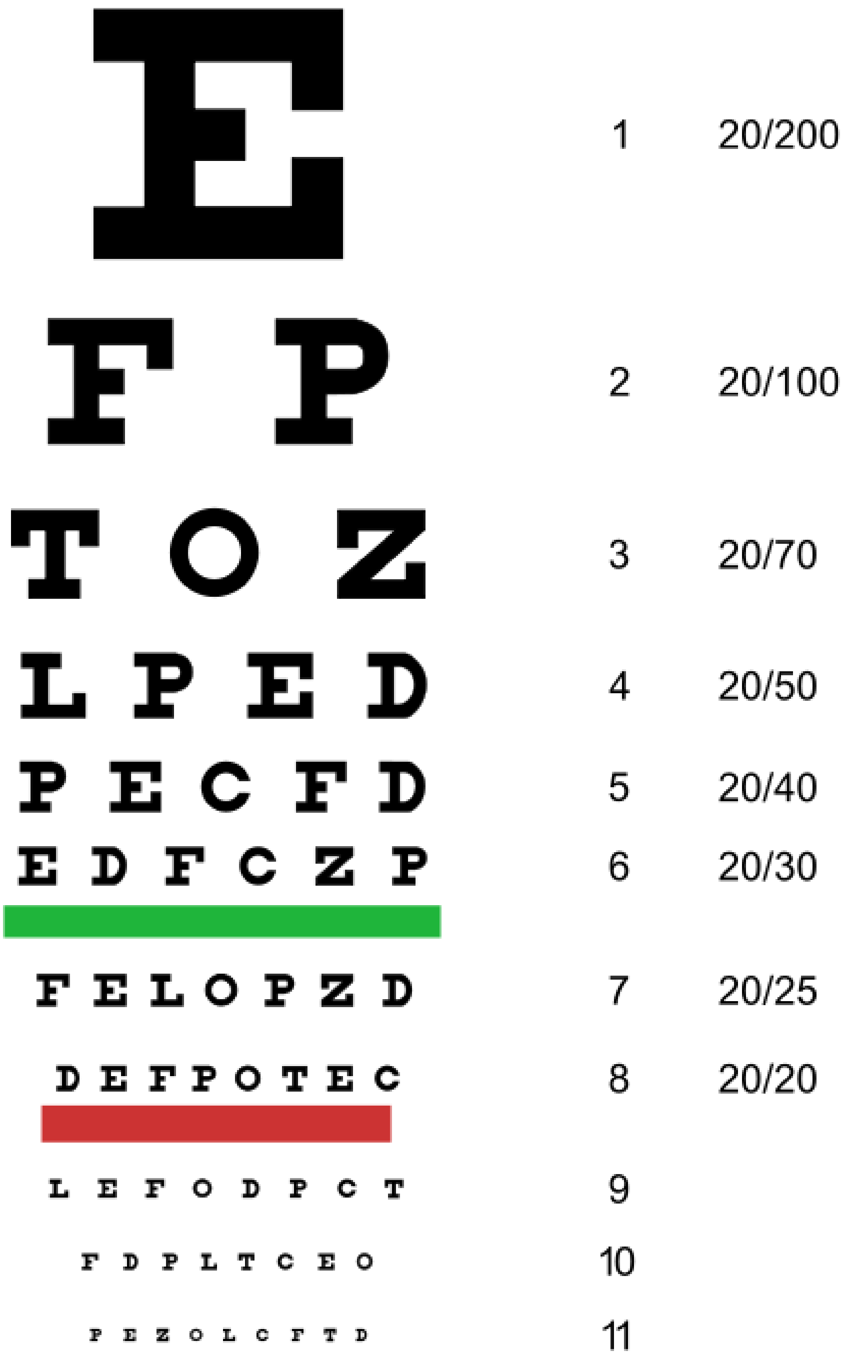
- ☐ Deuteranopia (Green-blind)
- ☐ Tritanopia (Blue-blind)
- ☐ Protanopia (Red-blind)

Indicate the preferred hand

	Always left	Mostly left	No preference	Mostly right	Always right
Writing	☐	☐	☐	☐	☐
Throwing a ball on a target	☐	☐	☐	☐	☐
Playing with a racket	☐	☐	☐	☐	☐
What hand is up to handle a broom	☐	☐	☐	☐	☐
What hand is up to handle a shovel	☐	☐	☐	☐	☐
Lighting matches	☐	☐	☐	☐	☐
Scissors when cutting paper	☐	☐	☐	☐	☐
To hold a wire to move it through the eye of a needle	☐	☐	☐	☐	☐
To deal playing cards	☐	☐	☐	☐	☐
To hammer in a nail	☐	☐	☐	☐	☐
To brush your teeth	☐	☐	☐	☐	☐
To remove a jar lid	☐	☐	☐	☐	☐
	-2	-1	0	1	2

Appendix C

Test chart for the visual acuity test



Appendix D

Informed consent

Informed Consent

On a voluntary basis, I decided to participate in an experiment in which response times and the accuracy of my choices will be measured.

The experiment consists of subtasks, including a Stroop task, the attentional blink paradigm, a useful field-of-view task, a delayed recognition task and a visual acuity test.

I have been informed about the specific purpose of the research will have the opportunity to ask further questions after the experiment. If I have additional questions later on, I can always contact the researcher Julian Steinke (j.steinke-1@student.utwente.nl).

I have the right to stop with the experiment at any time.

I understand that the data gathered in this experiment will be used for a thesis and might be published. My anonymity and the anonymity of my data is assured.

______________________________	______________________________
Name participant	Name researcher
______________________________	______________________________
Signature participant	Signature researcher

____

Place

____

Date