The Photographer's Guide to Los Santos

Contents

Preface **4**

1 Introduction **5**

About The Photographer's Guide to Los Santos 5

Grand Theft Auto V Studies 6

Grand Theft Auto V Tourism 6

Grand Theft Auto V Art Education 7

2 Architectural Photography **9**

The Continuous City by Gareth Damian Martin 17

Readings 17

Tutorial 17

Content Replication Assignment 18

Selected Student Projects 19

3 Social Documentary **20**

Down and Out in Los Santos by Alan Butler 20

Readings 21

Tutorial 21

Content Replication Assignment 21

4 Re-enactment Photography **22**

26 Gasoline stations in GTA V by Lorna Ruth Galloway 22

A Study on Perspective by Roc Herms 23

Further references 23
Readings 23
Tutorial 23
Content Replication Assignment 26

5 Nature Documentary **27**
Grand Theft Auto V's Botany by Ryohei Takahashi 27
San Andreas Streaming Deer Cam by Brent Watanabe 27
Readings 28
Tutorial 28
Content Replication Assignment 34

6 Surrealist Photography **35**
Alexey Andrienko aka HAPP v2 35
Readings 35
Tutorial 35
Content Replication Assignment 63

7 Hyperrealism **64**
Readings 65
Tutorial 65
Content Replication Assignment 70

8 Glitch Art **71**
Readings 71
Tutorial 71
Content Replication Assignment 91

9 Meta-Photography **92**
Readings 92
Tutorial 92
Content Replication Assignment 109

Preface

This space hosts the work in progress for the project *The Photographer's Guide to Los Santos*. Content on these pages is currently being added and edited, and none of it is to be considered final. All material is currently presented only for educational purposes and to be used solely in class by the author.

Chapter 1

Introduction

About The Photographer's Guide to Los Santos

The Photographer's Guide to Los Santos sits between a touristic guide and a photography manual, and between an exhibition catalogue and a peak behind the scenes of artwork creation.

The Photographer's Guide to Los Santos is an ongoing project that builds on top of a research on artistic practices within spaces of computer games, with a particular focus on in-game photography, machinima and digital visual arts. It follows some themes and ideas previously explored in the exhibition *How to Win at Photography*, while focusing more specifically on the relationship between computer games and photographic activities inside the world of Grand Theft Auto V.

The idea of a guide refers to in-game photography as a form of 'virtual tourism' (Book, 2003), which was also the premise of an actual tourist guide published by Rough Guides in their 2019 *Rough Guide to XBOX*. Yet this guide project also understands the game world as a site for image production and artistic creation, turning the game into a destination for a 'game art tourist'. The Photographer's Guide to Los Santos presents the game environment of Grand Theft Auto V both as a space to explore and in which to create images, as well as a place to navigate and learn about some of the most important artworks that it has enabled to create.

The project also brings together several experiences from teaching in-game photography as an artistic practice in different educational settings and institutions, compiling materials and tools for students and artists interested in engaging with the field. The tourist guide of the game world doubles as a photography manual for the in-game photography age, featuring tutorials ranging from game screenshotting to computer programming for creative modding. Through the

practical exercises, the project invites to rethink the game object as a space for creative, subversive and critical endeavours, which can be played differently, documented, reclaimed or modified through an artistic approach.

Finally, the project draws inspiration from the works of artists who have explored the 'metaplay' of photographing game worlds instead of following the game rules and attempt to reach the goal of winning. The Photographer's Guide to Los Santos is indebted to all the artists it features, but was particularly inspired by Gareth Damian Martin's live streamed workshop *Photography Tour of No Man's Sky* (realized for Now Play This Festival 2020), Total Refusal & Ismaël Joffroy Chandoutis's 2021 in-game lecture performance and guided tour *Everyday Daylight* (realized for the CCS Paris), and Alan Butler's epic 2020 live endurance performance *Witness to a Changing West* (realized for Screen Walks) and his 'Content Replication Assignments'.

Grand Theft Auto V Studies

Los Santos is the Grand Theft Auto V's fictional, parodic version of real-life Los Angeles. Just like Los Angeles is the global centre of film and commercial media production, Los Santos is the epicentre of in-game photography and machinima creation. While it may seem reductive to only focus on a single game to address the larger phenomenon of in-game photography, GTA V is the biggest source of creative outputs to date, with its extended open world and one of the largest community of active modders.

Launched in 2013, the game contains a world map of more than 80 square kilometers of total area, which includes the urban area of the city of Los Santos and the rural area of Blaine County. This incredibly vast environment features a large desert region, dense forest, several mountains, beachside towns, on top of the large metropolis of Los Santos. The game simulates the everyday life of hundreds of individual NPCs (while it allegedly counts a population of over 4 million) as well as counting 28 animal species, and more than 800 buildings in GTA V are based on real-life landmarks.

The size of the photorealistic simulation is only matched by the complexity of the game engine and its code, which - thanks to the effort of GTA V's modding community - allows players to use the game world as a powerful tool to create new scenes, take controls of its algorithmic entities, modify cameras and reshaping the game into a movie set or a photo studio.

Grand Theft Auto V Tourism

The project can be seen as something between a playful travel guide of Los Santos and one of the star maps offered to tourist in Hollywood, pointing to the

homes of movie actors and hollywood celebrities. This guide allows players to explore the game environment following some of the most interesting artworks that have been created with(in) it. It's divided in thematic chapters that follow different artistic practices, taking place in different locations of the game environment, followed by different tutorials and exercises connected with the works and the space analyzed.

The themes explore different approaches and practices connected to established artistic and photographic currents, with a general introduction text that gives an overview of the ideas, contexts and issues connected to the specific topic. A selection of artworks for each themes is presented by a curatorial statement, introducing the work and its artistic relevance. The work is accompanied by information on the in-game location in which it was produced, inviting the readers to reach the destination in Grand Theft Auto V through maps and indications.

The game environment thus becomes the space for possible 'art tours', getting insights into the artworks made in GTA V. This form of game tourism allows the player to see the behind the scenes, and experience the making of the works in its place of origin. While the complex algorithms of GTA V produce unexpected interactions and scenes, Los Santos is also stuck in the same time forever. Gas stations, shops, palm trees remain in the same state and location forever, allowing the tourist to witness the exact scene that was first encountered by the artists.

Grand Theft Auto V Art Education

This project is also an attempt to introduce a video game as a space for artistic intervention, and an invitation to use its mechanics, its code and its environment as a creative tool itself. The game can be played, documented and captured through a form of artistic play, that differs from normative gameplay and does not focus on advancing and winning but rather engages with the game object critically. Furthermore, the game software can also be manipulated, modified and used as an apparatus to create new images and interactions. The goal of this guide is to combine a curatorial approach that leads the viewer to discover the artworks made in GTA V with a hands on approach that teaches the player the tools for possible artistic interventions in this space.

Games are often seen as producing specific cultures and shaping identities through forms of play that follow the intentions of the developer. Here we understand games as objects to be reclaimed and tools to be deconstructed and rebuilt, both conceptually and literally. Consequently, players are not just passive actors that push buttons in the sequence that they are taught by the machine and its softwares, but open up the black box of the game and become critical thinkers and makers that actively play with the game, or even against it.

The Photographer's Guide to Los Santos can be employed as a resource to accompany workshops for students and artists approaching computer games and interested in learning how to engage with it. Each thematic chapter features a tutorial section that introduces different techniques and strategies to capture images within Grand Theft Auto V, connected to the examples and locations of each section. The chapters are thought to be experienced in order, as the tutorials at times rely on knowledge that is built on top of previous lessons. Each tutorial is accompanied by content replication assignments, in which the readers is invited to use the skills learnt from each chapter to recreate a work presented in that section. Tutorials are intended for anyone who might be playing GTA V for the first time in their life and do not assume any previous experience, although some basic idea of programming is helpful when dealing with scripting and modding the game.

Chapter 2

Architectural Photography

> "[…] a virtual urban life, like playing on the keyboard of the city as if it were a land of screen. I saw it as the end of architecture … by pushing the concept to its limit and primarily by using the photograph as a point of departure. This is reflected in the idea that the great majority of images are no longer the expression of a subject, or the reality of an object, but almost exclusively the technical fulfilment of all its intrinsic possibilities. It's the photographic medium that does all the work. People think they're photographing a scene, but they're only technical operators of the device's infinite virtuality. The virtual is the device that wants nothing more than to function, that demands to function." – Jean Baudrillard - *The Singular Objects of Architecture and Philosophy*, 2000

Architectural photography was born with the invention of heliographs, daguerreotypes and large format cameras in the first half of the 19th century. Due to the long exposure times, buildings were the ideal subjects for the early scientific experiments of Joseph Nicéphore Niépce, Louis-Jacques-Mandé Daguerre and William Talbot.

View from the Window at Le Gras by Joseph Nicéphore Niépce, 1826-27.

Paris' Boulevard du Temple by Louis-Jacques-Mandé Daguerre, 1839.

While the relationship between architecture and photography has been part of the medium from its birth for technical reasons, this form of image making has evolved to visually explore the connection with material spaces and forms, as well as the relation between human perception and architectural bodies. The photographic image is not simply a document of a structure, but "is, in fact, *part* of its architecture".[1] Curator Urs Stahel wrote that "pictures […]

[1] Lorenzo Rocha, *Photography and Modern Architecture* ,"Concrete - Photography and Architecture", Scheidegger & Spiess, 2013

offer a discourse that is unlike the physical experience of architecture. They transform volume into surface; distil matter into forms and signs. Photography shapes architecture, enlarging and reducing it, heightening and shortening it, accentuating it, yet largely leaving it to its own devices."[2]

The first architectural photographer is considered to be Joseph-Philibert Girault de Prangey, who started to take daguerreotypes of iconic buildings like the Parthenon in Athens and Notre-Dame in Paris from 1841. Architectural photography evolved in two distinct approaches, namely Elevation and Perspective. The Elevation Approach focuses on representing a structure as a two-dimensional image, obtaining a viewpoint that is parallel to the building and aimed at showing as many details as possible. The Perspective Approach aims at depicting the structure within the space, focusing on the third dimension and often taken at an angle or from a vantage point from a corner.

Facade and North Colonnade of the Parthenon on the Acropolis, Athens by Joseph-Philibert Girault Prangey, 1842.

Modern architecture started to become an increasingly popular subject within photography as the urban landscapes began to be reshaped. Around 1900, French photographer Eugène Atget started focusing on the disappearing architecture of "Old Paris". He captured the alleys and buildings of pre-revolutionary Paris, which were going to be demolished as part of a huge modernization project. Many of Atget's photographs were taken at dawn, which - combined with the urban solitude and emptiness he portrayed, created a special sense of space and ambience. Many photographers were greatly influenced by his images, including American photographer Berenice Abbott who bought most of Atget's negatives and prints before moving back to New York from Paris.

Coin rue de Seine, by Eugène Atget, 1924.

Back in New York, Abbott was confronted with a similar modernization process, with the old New York which was fast disappearing. "At almost any point on Manhattan Island, – she noticed – the sweep of one's vision can take in the dramatic contrasts of the old and the new and the bold foreshadowing of the future. This dynamic quality should be caught and recorded immediately in a documentary interpretation of New York City. The city is in the making and unless this transition is crystalized now in permanent form, it will be forever lost [...]. The camera alone can catch the swift surfaces of the cities today and speaks a language intelligible to all."[3]

Chanin Building, New York by Berenice Abbott, about 1935.

The modernist project in architecture meant embracing a more minimalist approach and rejecting ornament. A rational use of material was combined with

[2] Urs Stahel, Foreword to "Concrete - Photography and Architecture", Scheidegger & Spiess, 2013

[3] Abbott, Berenice, "Changing New York" (project proposal, New-York Historical Society, 1932); excerpted in O'Neal, Hank. Berenice Abbott: American Photographer. (New York: McGraw-Hill, 1982): 16–17.

an analytical and functional approach. Similarly, modernist photography rejected formal images and the painterly qualities of the pictorialist tradition in favour of a sharp focus, crisp lines and repetition. It celebrated the apparatus as a mechanical tool, and called for a "straight photography"[4]. The city played a major role in this relationship between urban spaces and their images, with radical architecture and modernist photography sharing an ideological connection. Architects and photographers started working as closely connected pairs in the second half of the XX century. Armando Salas Portugal for Luis Barragán, Bill Engdahl for Mies van der Rohe, Julius Shulman for Richard Neutra are some examples of this tightly connected relationship between the architecture and the image that shapes the space of the time.[5]

The tradition of architectural photography deeply informs in-game photography practices. Games often attempt to include realistic simulations of existing architecture and allow players to navigate their world through the game camera in a way that is directly linked to the camera operations of the analogue photographer who wants to document the city. Rockstar's Grand Theft Auto series has produced game versions of New York (Liberty City), Los Angeles (Los Santos) and Miami (Vice City). Ubisoft's Watch Dogs franchise modelled its worlds around Chicago, San Francisco and London. Actual photographs of architecture and urban spaces lie at the core of how buildings and streets are modelled and simulated in these spaces: the Watchdogs team "made repeated visits to take photos of different neighbourhoods" and Rockstar "distilled 250,000 photographs and countless hours of video into Los Santos, their version of Los Angeles and its hinterland."[6] While the games provide reduced approximations of their original counterparts, most of their landmarks are reproduced in great detail and accuracy. These digital copies and playable spaces can also be considered archival doubles for heritage architecture that might get damaged or disappear. When the cathedral of Notre-Dame de Paris suffered a structural fire beneath the roof in 2019, it left the iconic Gothic architecture with its spire collapsed, its upper walls severely damaged and most of its roof destroyed. Caroline Miousse, a level artist on the game Assassin's Creed Unity, took around two years modelling the cathedral inside and out, spending 80 percent of her time on the Notre Dame. Miousse "spent literally years fussing over the details of the building. She poured over photos to get the architecture just right, and worked with texture artists to make sure that each brick was as it should be."[7] At the time of the fire, many players and fans of the game recalled the 3D

[4]Sadakichi Hartmann, "A Plea for Straight Photography," American Amateur Photographer 16 (Mar. 1904), pp. 101–09; reprinted in Beaumont Newhall, ed. Photography: Essays and Images (Museum of Modern Art, New York 1980), p. 186.

[5]For more information about these three case studies see Lorenzo Rocha, *Photography and Modern Architecture* ,"Concrete - Photography and Architecture", Scheidegger & Spiess, 2013.

[6]Hoal, Phil. "From Watch Dogs to GTA V, why 'video games are going to reshape our cities' ". The Guardian, 10 June 214. https://www.theguardian.com/cities/2014/jun/10/watch-dogs-gtav-video-games-reshape-cities-sim-city-will-wright

[7]Webster, Andrew. "Building a better Paris in Assassin's Creed Unity", The Verge, 17 April 2019 (originally published in on 31 October 2014). https://www.theverge.com/2014/10/31/7132587/assassins-creed-unity-paris

model of the cathedral and speculated if the video game could provide help for the reconstruction plans. Ubisoft offered to provide the reconstruction effort with over 5,000 hours' worth of research on the structure, created relying on "pictures — photos, videos — of modern day Notre-Dame."[8]

Diagram showing the player navigation in Assassin Creed Unity's Notre Dame by Luke Caspar Pearson and Sandra Youkhana, in *Videogame Atlas: Mapping Interactive Worlds*, Thames & Hudson, 2022.

Yet the game is not just a simulation of the physical space and realism is more of an added visual effect, which is still subordinate to gameplay. The architecture in the game is still part of an environment to be played, and not just seen. In the case of Assassin's Creed Unity, this is especially true, as the game mechanics focus on the character's ability to climb over the roofs and buildings of the city. Gameplay priorities intervene on the game architecture, setting realism aside in favour of features that allow the player to better interact with the space. "We added things like cables and incense across the second level of Notre Dame so players would be able to move around easier when they're above the ground," Miousse explained on Ubisoft blog UbiBlog.[9] Windows that swing open on the upper levels of the cathedral have also been added, along with gilt panels on the balustrades of the tribune and along the nave, which guide the movements of the player. Bruce Shelley writes that realism contributes to a greater effect but is not essential in gameplay: "Realism and historical information are resources or props we use to add interest, story, and character to the problems we are posing for the player. That is not to say that realism and historic fact have no importance, they are just not the highest priority."[10] Speaking to The Verge, art director Mohamed Gambouz noted that the pointy rooftops of Paris in the game had to be smoothed out so as not to interfere with the player's parkour flow. That was just "one of many changes made to make ACU's Paris not only work better for a game, but also match the vision of Paris that players have in their mind. Gambouz calls it the "postcard" effect. 'When people talk about Paris they have postcards in their mind,' he says, 'even if this postcard isn't accurate or truthful to the setting.'" [11] There are also technical restrictions that get in the way of realism in game architecture. For example, the tympana above the main doors of Notre Dame's cathedral are full of intricate sculptures, which are not rendered as 3D objects in the game. Instead, the game designers created 2D textures that give the appearance of being sculpted , through "an optimization technique that looks great and doesn't compromise immersion — unless you

[8] Gilbert, Ben. "As France rebuilds Notre-Dame Cathedral, the French studio behind 'Assassin's Creed' is offering up its 'over 5,000 hours' of research on the 800-year-old monument", 18 April, 2019. https://www.businessinsider.com/notre-dame-fire-assassins-creed-maxime-durand-ubisoft-interview-2019-4?r=US&IR=T

[9] https://blog.siggraph.org/2019/05/how-ubisoft-re-created-notre-dame-for-assassins-creed-unity.html/

[10] Shelley, Bruce. "Guidelines for Developing Successful Games", Gamasutra, August 15, 2001. quoted in Galloway, Alexander R. *Gaming – Essays on Algorithmic Culture*, University of Minnesota Press, 2006

[11] https://www.theverge.com/2014/10/31/7132587/assassins-creed-unity-paris

get really close and swing the camera around to break the illusion."[12] Finally, the game architecture might also be affected by copyright laws that protect the original building. "Notre-Dame is […] owned by the French state, not by the Catholic Church, and it's a designated historic monument, to boot, one that's continually being restored. Much of the cathedral is under a patchwork of copyright restrictions."[13] That meant that all of the sculptures, the paintings, the rose windows of the church could not be authentic to real life and had to be adapted by the designers to circumvent French copyright laws. Miousse spoke of Notre-Dame's organ: "It's just so huge and beautiful… and copyrighted. We couldn't reproduce it exactly, but we could still try to nail the feeling you get when you see it." [14]

Similarly, Grand Theft Auto V's Los Santos is a parodic approximation of 2011 Los Angeles, which "is selectively shrunk for the experience of the gamer. Los Santos represents Los Angeles through fragmentation, simplification and social stereotypes."[15] Countless online videos and articles celebrate Los Santos' realism, with a whole genre of "GTA 5 vs Real Life" posts comparing game screenshots with photographs of Los Angeles and its landmarks. The graphical qualities of photorealism of Grand Theft Auto V and the painstaking digital doubling of buildings and structures leave no doubt in each of these posts that Los Santos represents Los Angeles. Its toponymy doubles down on the copy: the Griffith Observatory appears as Galileo Observatory, The Bixby Bridge in Big Sur becomes the Big Creek Bridge, Pershing Square is Legion Square and the iconic Hollywood sign is transformed into the same signage that reads Vinewood instead. Mark D Teo notes that "the Los Santos conceptualisation of modern Los Angeles is omnipresent and mocking; it is an extension of its former city."[16] Grand Theft Auto V is certainly not the first mediated representation of the city of Los Angeles, which has appeared so often in film and television to create in its visitors a sense of uncanny familiarity. D. J. Waldie writes that "we are all citizens of Los Angeles because we have seen so many movies"[17] and critic Keith Stewart has suggested that the movie Drive is essentially "a non-interactive version of Grand Theft Auto."[18] Will Jennings argues that Los Santos is to be interpreted as a Situationist fragment collage of Los Angelese, "akin to Guy Debord's Naked City (McDonough, 2002, p.241), using ideas of the dérive and utilising simulated landmarks not only as mediators between

[12] https://www.polygon.com/features/22790314/assassins-creed-unity-notre-dame-restoration-accuracy

[13] ibid.

[14] https://blog.siggraph.org/2019/05/how-ubisoft-re-created-notre-dame-for-assassins-creed-unity.html/

[15] Teo, Mark D. "The Urban Architecture of Los Angeles and Grand Theft Auto", 2015, p.5. https://www.academia.edu/18173221/The_Urban_Architecture_of_Los_Angeles_and_Grand_Theft_Auto

[16] ibid. p.50

[17] Waldie, Donald J. "Beautiful and Terrible: Los Angeles and the Image of Suburbia", in *Seeing Los Angeles: A Different Look at a Different City.* Los Angeles: Otis Books. 2007

[18] *How Video Games Changed the World.* Directed by Al Camborn, Marcus Daborn, Graham Proud & Dan Tucker (film). London: Channel 4. 2013. 1hr8mins

simulated space and the real L.A. but also as navigational devices as outlined in Kevin Lynch's 1960 Image of the City (Tea, 2005)."[19]

Screenshot of Lo Santos' Vinewood sign in Grand Theft Auto V.

Game architecture is therefore not simply a copy of its physical counterpart and it's not only gameplay, technical limitations and copyright that affect virtual buildings and spaces. Furthermore, the digital technologies at the core of computer graphics and games have completely transformed architecture and photography, as well as the relation between the two. The image in its digital and networked form has come to shape the experience of architecture, to the point of even affecting its creation. Designer Alexandra Lange, talking to the podcast 99% Invisible, says that especially hotel and restaurant interiors are being affected by Instagram. These spaces are being increasingly designed to lure visitors with their Instagrammabilty. Lange recalls: "You know, one architect told me that it's really important to have an Instagrammable bathroom."[20] There is a temporal reversal between the architectural photography of the XIX and XX century that documented the experience of the space and contemporary spaces that are built by the image that they will create and distributed online. The image, in other words, comes first. Similarly, architectural renderings of newly planned neighbourhoods create an image of the future and idealised architecture. They are spaces that do not yet physically exist and where the architectural photograph not only precedes them but create them. Before they are built, these spaces are constructed through CGI and software that are enmeshed in both the architectural process as well as the imaging systems. Joel McKim argues that architectural renders belong to the category of "projective images", as they "attempt to bring an as yet non-existent reality into being."[21] Such images are becoming ubiquitous and ever more powerful, employed in visualisations of urban speculations and simulation technologies. For McKim these projective images are very much connected to Flusser's idea of the technical image: "while traditional images depict or represent the outside world, technical images envision or inform it. Technical images, in other words, are not mirrors but projectors. And what they project outwards, in very circular fashion are our very own models, concepts and texts – what Flusser calls 'programs'. Technical images, we might say, are not images or representations of the world, they are visions of the world remade to correspond to our own texts, concepts or programs. Architectural renders are, in many ways, quintessential examples of the technical images Flusser had in mind."[22] For Tobias Revell, these architectural images "overwhelm us into an unquestioned sense that this is the future", through "their aesthetic richness – usually highly saturated, dra-

[19] Jennings, Will. "Learning from Los Santos: Computer games & their effect on our reading of space", 2018. https://willjennings.info/Essay-Learning-from-Los-Santos-Computer-games-their-effect-on-our

[20] https://99percentinvisible.org/episode/instant-gramification/

[21] McKim, Joel. "Into the universe of rendered architectural images", Unthinking Photography, 2019. https://unthinking.photography/articles/into-the-universe-of-rendered-architectural-images

[22] ibid.

matic and glossy – as well as their physical size [...]. Once again, they piggyback the camera's perceived objectivity but this time into a kind of hyper-real fantasy so convincing that any disbelief is suspended."[23] Similarly, game design has been influenced by the history of architecture while at the same time game environments have affected architects, "blurring the boundaries of what belongs to the physical or the virtual world, with principles from both sides transferring either way."[24]

Game space, however, sits between the copy of physical structures and architectural renderings: it is "a space neither separate from real space nor simply a continuation of it."[25] Game worlds are somehow stuck between simulating physical spaces in a way that is as realistic as possible, while shaping new visions of cities and "projecting" an own image of a world. They are made of (and from) photographs, and they generate new images through computational processes and realistic graphics. Gareth Damian Martin notes that "we are outnumbered by virtual worlds, overwhelmed by virtual architecture. Videogames and digital art have furnished us with a hundred thousand matterless forms– landscapes where no rock or earth has ever been present, cities founded on depthless skins of image and texture, expanses that will never see the light of a true sun. And yet, somehow there is material here, a new kind of matter. Some of it is borrowed—photographs, texture references, photogrammetry. Other parts are inherent properties of digital worlds— their obsession with surface, the logic of their light, their base particles; pixels, voxels, polygons."[26] On top of that, game spaces are also made to be navigated and experienced. The player who explores the built space inside computer games might turn into a photographer. They stop playing to screenshot virtual architecture, to study the game space in relation to the player.

In-game photographers might also investigate virtual architecture from the perspective of its materiality – its textures and 3D models – exploring how its structures are "built". In his series *The Edge of The World* and *Flying and Floating*, Robert Overweg examines the impossible buildings and the incongruous textures that make the architectures of games like Half Life 2 and Left 4 Dead 2. The artist writes: "I try not to follow the roads I am supposed to take, but try to seek out my own path within and outside the given boundaries of the game. I find joy in making use of a glitch/error which gives me the possibility to have a different look at the virtual world. Flying around and running through walls which I am not supposed to do gives me a sense of freedom and the ability to move in ways I can't in the physical world. I want to look behind the curtain of the virtual facade and show it to the world."[27] Architectural glitch photog-

[23]Revell, Tobias. "Rendering the Desert of The Real", Unthinking Photography, 2019. https://unthinking.photography/articles/rendering-the-desert-of-the-real

[24]Foulston, Marie. "Introduction", *Videogame Atlas: Mapping Interactive Worlds.* Luke Caspar Pearson and Sandra Youkhana. Thames & Hudson. 2022. p.7

[25]Cohen, Julie E. "Cyberspace As/And Space". *Georgetown Law Faculty Publications and Other Works. 807.* 2007. scholarship.law.georgetown.edu/facpub/807

[26]Martin, Gareth Damian. "Represented , Contested, Inverted". in *Heterotopias 001.* 2017

[27]https://www.shotbyrobert.com/glitches

raphy is visually reminiscent of Gordon Matta-Clark's 1970s "anarchitecture" works. These were photographs and films documenting the artist sawing and carving sections of buildings, revealing the infrastructure and hidden materials of architectural structures.

From the series *Flying and Floating* by Robert Overweg.

Alternatively, photographers use the game object as a tool itself, a device to create structures and architectural images. In a course on digital and networked photography held at F+F School for Art and Media Design Zurich in 2019 (which among other experiences was an inspiration to The Photographer's Guide to Los Santos), Kaja Fuchs worked with her younger sister to create buildings in the game Minecraft based on the iconic work of German photographer Hilla and Bernd Becher. For forty years, the german duo photographed disappearing industrial structures like water towers, coal bunkers, gas tanks and factories. Their images were always taken in black and white and the photographers approached every building in an almost scientific way, bringing multiple structures together in grids. The Fuchs sisters acted in a similar connection to the modernist architect and photographer pairs of the second half of the XX century, reconstructing the water towers in the 3D game and controlling the camera and the visual effects to reproduce the strict photographic rules originally employed by the Bechers. The game space allowed the players to act as both architect and photographer.

A reinterpretation of Hilla and Bernd Becher's *Water Towers* in Minecraft by Kaja Fuchs.

Architecture and photography have been interacting with each other for almost two centuries, affecting each other and taking different roles in many processes of building spaces and imag(in)ing structures. The architectural photograph was born as a representation of the past and from the distance. Throughout the XX century architects and photographers would shape the present by shaping spaces and creating images following the modernist ideology. In its computational and networked form, the image begins to copy physical structures and analogue architectural photography but soon proceeds to shape the creation of space itself. Through its computational simulation, photorealistic graphics and networked circulation, it imagines and projects onto the world. It precedes the built space in the physical world. This virtual architectural image is finally able to be navigated, documented, shaped and played once it becomes part of the game world. Players turned photographers and artists are entangled in this complex journey where images and structures constantly move closer and further apart, where new and old materials merge and structures reject or submit to the laws of physics. In-game photographers document the game architecture but also investigate it, actively explore it and even shape it.

The Continuous City by Gareth Damian Martin

Gareth Damian Martin, *Outskirts*, from *The Continuous City*,

Gareth Damian Martin, *Pathways*, from *The Continuous City*,

artwork text

More about *The Continuous City*

Interview with Gareth Damian Martin

Getting there

- The intersection of Interstate 4 and Interstate 5 manifests the architecture of traffic of the megalopolis.

Readings

Heterotopias

Mark D Teo, The Urban Architecture of Los Angeles and Grand Theft Auto, 2015.

Tutorial

Playing as a photographer

Playing as a photographer means we are going to skip the compulsory game introduction, the missions and the narrative plot of the game. We are going to install a file which allows us to travel to Los Santos with no restriction of any kind, so that we can focus on photographic activities.

To do this we use a completed game save file. Download the file and extract its content, then copy both files `SGTAXXXXX` and `SGTAXXXXX.bak` in `Documents/GTA V/Profiles/YYYYYYYY/`. Once the files are there, you will be able to load the game state by running GTA V and navigating on the pause menu to `GAME` > `Load Game` and select the 100% game file.

With the game save file loaded the player is free from the constraint of the main game missions and locked map areas, but the game will still at times resist the photographic project in different ways. The player can be attacked and killed by mountain lions, or beaten up and sent to the hospital by non playable characters. They can drown, fall to their death and get run over by cars.

Photographing the Game Screen

Analogue Game Photography

//To do: technical setup and moiré effect, artists using cameras rather than screenshotting, idea of separating the photographic act "of" the screen rather than the screen copy of the screenshot… //Photographs of a TV screen taken with a digital camera often exhibit moiré patterns. Since both the TV screen and the digital camera use a scanning technique to produce or to capture pictures with horizontal scan lines, the conflicting sets of lines cause the moiré patterns. To avoid the effect, the digital camera can be aimed at an angle of 30 degrees to the TV screen.

Screenshotting

On windows there are several ways to take a screenshot. To capture your entire screen and automatically save the screenshot, press the `Windows logo key` + `PrtScn key` (or `fn key` + `Windows logo key` + `PrtScn key`). The screenshot will be saved to `Pictures > Screenshots folder`.

//Steam

On windows 10 and 11 you can use the Game bar to take game screenshots and start/stop game screen recordings. Press the `Windows logo key` + `G` on your keyboard to open Game Bar.

- Press the camera icon to take a screenshot of the game screen.
- Press the circle icon to start a clip, then the square icon to stop recording the game screen.
- Click on "See my captures" to access the image and video files.

Content Replication Assignment

Drive around Los Santos or go to the location on the map and take pictures of the architecture of the city through photographs or screenshotting. You are free to try to recreate the images of Gareth Damien Martin or explore different areas and approaches related to architectural photography, virtual urban spaces, digital structures and other topics related to the explored themes.

The Photographer's Guide To Los Santos Workshop At Camera Arts, Lucerne University Of Applied Sciences And Arts, January 2023

Selected Student Projects

2.0.1 *Luftige Enge* by Mascha Negri {–}

Mascha Negri, from *Luftige Enge*, F+F, March 2023.

In *Luftige Enge* Mascha Negri focuses on the industrial areas of Los Santos. Wandering through the buildings in the game, the photographer looked for angles and perspectives that are reminiscent of the work of Germaine Krull as well as Bernd and Hilla Becher. Upon closer inspection, the images show the pixels of the screen, the polygons that make the 3D architecture and the textures of virtual pipes and chimneys. Negri worked with the smartphone camera to photograph the game screen, highlighting the properties of the simulation, while simulataneously following the trajectory of black and white architectural images from the analogue photography tradition. Negri employed digital editing through the smartphone photo app, using high-contrast black and white post-processing to underline the geomatric shapes of the building. The title *Luftige Enge*, literally "airy narrow", follows the contrast between the airy appearance of the smoke coming from the pipes and the massive size and dense positioning of the buildings. Finally, the contrast between analogue tradition and digital images, and between air and mass, can be extended to the relation the between post-industrial society and economy of the game industry and the vanishing heavy industry in the west simulated in the game buildings.

Mascha Negri, from *Luftige Enge*, F+F, March 2023.

Mascha Negri, *Luftige Enge*, F+F, March 2023.

Chapter 3

Social Documentary

//general intro on simulating society, the creation of NPCs, documentary and street photography traditions connected to politics of visibility and representation, and how they relate to the politics of simulation, how the player-photographer documents the creation of complex social spaces and reveals the process of simulating people and issues of class, gender, race in the game space…

//photo reporter in virtual worlds Marco Cadioli / Marco Manray reporter in Second Life net art manifesto / Arenae "like a modern Capa" and Roc Herms in Postcards from Home with virtual world of Playstation Home

Fear and Loathing in GTA V by Morten Rockford Ravn

More about *Fear and Loathing in GTA V*

//Yumo Wu, Refusal

//Wuji, chinese immigrant NPCs in RDR2

Down and Out in Los Santos by Alan Butler

artwork text

//the photographer showing the developer's biases?

More about *Down and Out in Los Santos*

Getting There

The homeless camp in Los Santos is under the Olympic Freeway in Strawberry.

Dignity Village is a tent city established by homeless people near Procopio Beach, east of Paleto Bay.

Getting There

Readings

Tutorial

In-game Smartphone Camera

Snapmatic is the photo app on your simulated mobile phone in GTA V.

- Press `UP` on the on the keyboard (PC) or d-pad (Playstation) to bring up your phone.
- Select the Snapmatic app - it's on the bottom left of the homescreen.
- Move the camera with the `Mouse` on PC, or with the `RIGHT STICK` on Playstation.
- Zoom in and out with the `Mouse Wheel` on PC, or `LEFT STICK` on Playstation.
- You can shuffle through filters with `DOWN` or borders with `UP`.
- To take selfie press the `Mouse Wheel Button` on PC or `R3 STICK` on Playstation to turn the camera on yourself.
- Once you're happy, take the photo with "È`nteron pc or`X"‘ on the Playstation. Press it again to save it to the Gallery.
- You can upload your picture to your Rockstar Game sSocial Club profile by going to the gallery and pressing the "Left Ctrl" key on PC.
- Your photos will be published on socialclub.rockstargames.com/member/USERNAME/photos, where USERNAME is replaced by your actual username.

Content Replication Assignment

Investigate the population of NPCs in Los Santos and explore the way specific people are simulated and represented. Wander atound Los Santos and Blaine County and take pictures of a specific community, reflecting on the politics of simulation and representation.

Chapter 4

Re-enactment Photography

//general introduction on the development of photorealism in games, the relationship between photography and CGI, the remediation of photographic images and the analog apparatus, the player as photographer situated in the tradition of conceptual photographers like Sherrie Levine and Sturtevant, the copy as a conceptual approach that create new meaning through a similar image but a different context…

26 Gasoline stations in GTA V by Lorna Ruth Galloway

artwork text

More about *26 Gasoline stations in GTA V*

Getting There

- Globe Oil Gas Station, Innocence Blvd & Alta St, South Los Santos
- LTD Gas Station, Davis Ave & Grove St, South Los Santos
- LTD Gas Station, Mirror Park Blvd & W Mirror Dr, Mirror Park
- Globe Oil Gas Station, Clinton Ave & Fenwell Pl, Vinewood Hills
- Xero Gas Station, Strawberry Ave & Capital Blvd, South Los Santos
- RON Gas Station, Davis Ave & Macdonald St, South Los Santos
- Xero Gas Station, Calais Ave & Innocence Blvd, Little Seoul
- LTD Gas Station, Lindsay Circus & Ginger St, Little Seoul
- RON Gas Station, N Rockford Dr & Perth St, Morningwood
- Xero Gas Station, Great Ocean Hwy, Pacific Bluffs

A Study on Perspective by Roc Herms

artwork text

More about *A Study on Perspective*

Getting There

Vinewood Sign, Vinewood Hills

Further references

Little Books of Los Santos by Luke Caspar Pearson

More about *26 Gasoline stations in GTA V* by M. Earl Williams

Readings

Tutorial

Director Mode

Director Mode allows players to interact with GTA V as a movie set where one can customize the character appearance, weather conditions, vehicle and pedestrian density, and much more. It was first introduced in the PC version of the game with the Rockstar Editor, and later added to the console title in 2015.

There are 3 ways to access Director Mode:

1. On the game pause menu go to `Rockstar Editor` tab (last tab on the right) and select `Director Mode`
2. press `M` key on PC to bring up the `Interaction Menu` and select `Director Mode`
3. Press the `UP Arrow Key` to take out the in-game smartphone and dial `1999578253` to activate `Director Mode`.

Note: Director Mode is unavailable when player is in a vehicle, wanted, or active in a mission.

Once activated, the game will open up the main Director Mode scene showing a casting trailer with a menu from which the player can choose from a number

of options. Here the player can change the appearnce of their character by selecting a different model.

The player can also set the scene, manipulating the conditions of the game world as if it were a movie set: time of day, weather conditions, populations and vehicle density can all be controlled from the Director Mode menu.

Here is the full list of options available in the Director Mode casting trailer menu:

Menu Item	Values
Actors	Animals, Beach, Bums, Costumes, Downtown, Emergency, Services, Gangs, Heist Characters, Laborers, Military, Online Characters, Professionals, Special Characters, Sports, Story Characters, Transport, Uptown, Vagrants
Time of Day	Midnight, Pre-Dawn, Dawn, Morning, Midday, Afternoon, Sunset, Dusk
Weather	Clear, Broken, Cloud, Overcast, Hazy, Xmas (snow), Smog, Fog, Rain, Thunder
Wanted Status	Normal, Low, Medium, High, Disabled
Pedestrian Density	Normal, Low, Medium, High, None
Vehicle Density	Normal, Low, Medium, High, None
Restricted Areas	On/Off
Invincibility	On/Off
Flaming Bullets	On/Off
Explosive Bullets	On/Off
Explosive Melee	On/Off
Super Jump	On/Off
Explosive Bullets	On/Off
Super Jump	On/Off
Slidey Cars	On/Off
Low Gravity	On/Off
Clear Area	On/Off

Once the player has selected their actor and scene setup, they can click on `Enter Director Mode` to launch the actress into the game worlld. Note: first person view is only available if the three protagonists characters are chosen.

In Director Mode players can quick travel to a pre-defined location across the game world. Create a waypoint on the `MAP` from the main menu by selecting a

destination on the map and press ENTER. Leave the menu and back in Director Mode press the M key to bring up the Director Mode main menu, and under Location press the LEFT ARROW KEY to select Waypoint, and press ENTER.

Scene Creator

Scene Creator is a functoin within Director Mode that allows players to place props, and save the created designs and layouts.

When in Director Mode press the M key on PC to bring up the Interaction Menu and select Director Mode > Scene Creator > Props > Rocks and Trees. Try choosing some trees and place them in the scene.

Capturing Gameplay

There are two ways of to capture gameplay: Manual Recording and Action Replay.

MANUAL RECORDING Hold ALT + F1 for Manual Recording. Gameplay will be recorded in the background until LEFT ALT + F1 is pressed to save the recording or LEFT ALT + F3 to cancel the recording.

ACTION REPLAY This option records gameplay footage after the fact. Press F2 to buffer recorded data in the background, but note it will not record anything until you press ALT + F1 to save action replay.

Recorded clips will be between 30 and 90s seconds in length, depending on how busy your action is. You will be notified if you are running out of space on your hard drive and you can define the amount of hard drive space you use for the Rockstar Editor by going to the game main menu in SETTINGS > ROCKSTAR EDITOR.

Rockstar Editor

On the game pause menu go to Rockstar Editor tab (last tab on the right) and choose Create New Project. Then select Add Clip to select the footage you want to use.

Recordings will be placed in a timeline: you can scroll through each clip using LEFT CTRL + X or Mouse Click and Drag them. Delete a clip using DEL, duplicate a clip with LEFT CTRL + C, or use the Mouse Wheel to scroll through the timeline.

Double Click on a clip or press ENTER to begin editing it.

You can change camera angles, effects, speed etc using these. You can scrub through the clip footage using the `LEFT Arrow Key` and `RIGHT Arrow Key` or the `Mouse Wheel`.

The Camera function allows players to position and control the game camera more freely, with less restrictions than the camera options in game play. There is an exception for gameplay footage captured in first-person perspective: here you cannot change camera angle. If there are any other camera restrictions, you'll see a notification on screen. Aside from that, three different camera settings are available in ROckstar Editor:

1. `Game Camera`
   This is the default third-person camera view filmed from behind the main character.
2. `Pre-set Camera angles` These are pres-set camera angles that lock the view at the front, behind, sides or above the main character. You can zoom in an out from these.
3. `Free Camera` The Free Camera can be placed within a defined area but freely around your main character.

Press `F5` to save your project and camera position.

Take a screenshot or export a Snapmatic picture by pressing the `TAB` key. This will save your picture in the game gallery and can only be exported out of the game by uploading the image to Rockstar Social Club.

Use the Game Bar to record a video or export your video using the `Export` option on the Project Main Menu. Here you can set frame rate and bit rate. The video will appear in the Video Gallery inside the Rockstar Editor main menu. From there you can view it and upload it to either YouTube or the Rockstar Social Club.

Scene Director Mod

//installation and setup

//setup mode (set the scene)

//active mode (record scene)

Content Replication Assignment

Chapter 5

Nature Documentary

//general introduction about the creation of a synthetic forms of nature, ecological issues, creation of virtual sublime, flora and fauna that are usually props that become the focus of the player's explorations, "virtual world naturalism"…

Grand Theft Auto V's Botany by Ryohei Takahashi

artwork text

More about Grand Theft Auto V's Botany_

San Andreas Streaming Deer Cam by Brent Watanabe

artwork text

More about *Deercam*

Getting There

Mount Chiliad is located in the Chiliad Mountain State Wilderness, and it is the tallest mountain in the game at 798m above sea level. The state park is home to lots of wildlife such as deer and mountain lions.

Getting There

Readings

Tutorial

Scripting Introduction

Preparation and Setup

- Install Windows 11
- Download and install Steam (with a copy of GTA V or buy the game if you do not have it. GTA V is 100+ GB so it will take a few hours depending on your internet connections)
- Download Script Hook V, go to the bin folder and copy `dinput8.dll` and `ScriptHookV.dll` files into your GTA V directory `C:\Program Files (x86)\Steam\steamapps\common\Grand Theft Auto V`
- Download Script Hook V dot net, copy the `ScriptHookVDotNet.asi` file, `ScriptHookVDotNet2.dll` and `ScriptHookVDotNet3.dll` files into your GTA V directory `C:\Program Files (x86)\Steam\steamapps\common\Grand Theft Auto V`
- Create a new folder in GTA V directory and call it "scripts".
- Download and install Visual Studio Community (free version of VS). Open Visual Studio and check the .NET desktop development package and install it
- Run GTA V and test if Script Hook V is working by pressing `F4`. This should toggle the console view. Try to type Help() and press "ÈNTER"' to get a list of available commands.

Creating a Mod File

- Open Visual Studio
- Select File > New > Project
- Select Visual C# and Class Library (.NET Framework)
- Give a custom file name (e.g. moddingTutorial)
- Rename public class Class1 as "moddingTutorial" in the right panel Solution Explorer and click Yes on the pop-up window

- In the same panel go to References and click add References...
- Click on > Browse > browse to Downloads
- Select ScriptHookedVDotNet > `ScriptHookVDotNet2.dll` and `ScriptHookVDotNet3.dll` and add them
- Go to > Assemblies and search for "forms" and select `System.Windows.forms`
- Search for "drawing" and select `System.Drawing`
- In your code file add the following lines on top:

```
using GTA;
using GTA.Math;
using System.Windows.Forms;
using System.Drawing;
using GTA.Native;
```

- Modify class moddingTutorial to the following:

```
namespace moddingTutorial
{
  public class moddingTutorial : Script
  {
      public moddingTutorial()
      {
          this.Tick += onTick;
      this.KeyUp += onKeyUp;
      this.KeyDown += onKeyDown;
      }

      private void onTick(object sender, EventArgs e)
      {
      }

      private void onKeyUp(object sender, KeyEventArgs e)
      {
      }

      private void onKeyDown(object sender, KeyEventArgs e)
      {
        if (e.KeyCode == Keys.H)
        {
              Game.Player.ChangeModel(PedHash.Cat);
          }
```

```
        }
    }
}
```

- Save file
- Go to Documents > Visual Studio > Project > moddingTutorial > moddingTutorial > `moddingTutorial.cs`
- Copy the .cs file in the GTA V directory inside the scripts folder
- Open GTA V, run the game in Story Mode (mods are only allowed in single player mode, not in GTA Online) and press 'H' to see if the game turns your avatar into a cat
- Note: every time you make changes to your .cs file in the scripts folder you can hit `F4` to open the console, type `Reload()` in the console for the program to reload the script and test again the changes.

onTick, onKeyUp and onKeyDown

The main events of Script Hook V Dot Net are onTick, onKeyUp and onKeyDown. Script Hook V Dot Net will invoke your functions whenever an event is called.

The code within the onTick brackets is executed every interval milliseconds (which is by default 0), meaning that the event will be executed at every frame, for as long as the game is running.

```
private void onTick(object sender, EventArgs e)
{
        //code here will be executed every frame (or per usef defined interval)
}
```

If your function is written inside onKeyDown (withiin the curly brackets following onKeyUp(object sender, KeyEventArgs e){}), your code will be executed every time a key is pressed. If your function is written inside onKeyUp, your code will be executed every time a key is released.

```
private void onKeyUp(object sender, KeyEventArgs e)
{
      //code here will be executed whenever a key is released
}

private void onKeyDown(object sender, KeyEventArgs e)
```

```
{
      //code here will be executed whenever a key is pressed
}
```

We can specify which code is executed based on what keys are pressed/released

```
private void onKeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.H)
    {
        //code here will be executed whenever the key 'H' is pressed
    }
}
```

Change Player Model

The player character is controlled as `Game.Player`. `Game.Player` can perform different functions, including changing the avatar model.

Change the 3D model of your character by using the `ChangeModel` function. The function needs a model ID, in order to load the model file of our game character. You can browse through this list of models to find the one you want to try (note: not all models seem to load properly).

These models are all PedHashes, basically ID numbers within the PedHash group. Copy the name of the model below the image and add it to PedHash. For example if you choose the model Poodle, you'll need to write `PedHash.Poodle`.

To change the model of your player character into a poodle you can write the following function:

```
Game.Player.ChangeModel(PedHash.Poodle);
```

add it in your .cs file in the onKeyDown event, triggered by the pressing of the 'h' key:

Example code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GTA;
```

```
using GTA.Math;
using System.Windows.Forms;
using System.Drawing;
using GTA.Native;


namespace moddingTutorial
{
    public class moddingTutorial : Script
    {
        public moddingTutorial()
        {
            this.Tick += onTick;
            this.KeyUp += onKeyUp;
            this.KeyDown += onKeyDown;
        }

        private void onTick(object sender, EventArgs e) //this function gets executed
        {
        }

        private void onKeyUp(object sender, KeyEventArgs e)//everything inside here is
        {
        }

        private void onKeyDown(object sender, KeyEventArgs e) //everything inside here
        {
            //when pressing 'H'
            if(e.KeyCode == Keys.H)
            {
                //change player char into a different model
                Game.Player.ChangeModel(PedHash.Poodle);
            }
        }
    }
}
```

Try to select different models and assign them to different keys to change the model of your character. Use keys that are not already implemented in the game controls to avoid clashes with built in operations.

Tasks

Our character can be controlled by our script, and given actions that override manual control of the player. These actions are called *Tasks* and in or-

der to assign tasks to our characters we have to define our `Game.Player` as `Game.Player.Character`. The `Game.Player.Character` code gets the specific model the player is controlling.

Now we can give tasks to the character by adding the `Task` function: `Game.Player.Character.Task`.

Finally we can specify what task to give the character by choosing a task from TaskInvoker list of possible actions.

Jump:

```
Game.Player.Character.Task.Jump();
```

Wander around:

```
Game.Player.Character.Task.WanderAround();
```

Hands up for 3000 milliseconds:

```
Game.Player.Character.Task.HandsUp(3000);
```

Turn towards the camera:

```
Game.Player.Character.Task.TurnTo(GameplayCamera.Position);
```

Some of the tasks are temporary and accept a time parameter (in milliseconds). Others are persistent, meaning they will keep being executed until the task is actively stopped. To stop a task you can use the `ClearAllImmediately();` command:

```
Game.Player.Task.ClearAllImmediately();
```

Task Sequences

You can create sequence of multiple tasks by using `TaskSequence` and the `PerformSequence` function. Create a new `TaskSequence` with a custom name, add tasks to it with `AddTask`, close the sequence with `Close` and then call `Task.PerformSequence` to perform the sequence.

```
TaskSequence mySeq = new TaskSequence();
mySeq.AddTask.Jump();
mySeq.AddTask.HandsUp(3000);
mySeq.Close();

Game.Player.Character.Task.PerformSequence(mySeq);
```

Random

We can add randomness by using a randomly generated number, which makes things outside of thepredefined programme controlled by us and introduces more autonomous behaviours. We use the `Random`function to create a randomly generated number between our minimum and maximum parameter (if only one parameter is inserted, the minimum is 0).

```
Random rnd = new Random();
int month = rnd.Next(1, 13); // creates a number between 1 and 12
int dice = rnd.Next(1, 7); // creates a number between 1 and 6
int card = rnd.Next(52); // creates a number between 0 and 51
```

Let's create a number to generate a random duration between 1 and 6 seconds, for the `HandsUp` task.

```
Random rnd = new Random();
int waitingTime = rnd.Next(1, 7);
Game.Player.Character.Task.HandsUp(waitingTime * 1000);
```

Subtitles and Notifications

Generate subtitles with a custom text string and duration (in milliseconds):

```
UI.ShowSubtitle("Hello World", 3000);
```

Generate a notification with a custom text string:

```
UI.Notify("Hello World");
```

Content Replication Assignment

Deercam reenactment

Write a mod script to change your game character into a deer by pressing a key, and make it autonomously wander around Los Santos by pressing another key.

Chapter 6

Surrealist Photography

//general introduction about avant garde traditions of distancing from reality and exploring the possibilities of CGI decoupled from realism adn life-like simulation, the game as an engine that can be used to create oniric scenes, which in turn reveal the untapped possibilities hidden within the game code, the player as a modder which can generate worlds within the world…

Alexey Andrienko aka HAPP v2

artwork text

More about Happ v2

Getting There

Chumash Beach

Readings

Tutorial

Scripting Characters

NPCs

NPCs are non playable characters and in GTA V scripting they are called `Peds`. Peds are an entity like Props or Vehicles and can be created, assigned different

model textures, equipped with weapons and controlled through different tasks.

Spawn a new NPC

A GTA V Ped can be created by the `World.CreatePed` function. This takes two parameters: an ID to assign the 3D model and textures, and the location where the Ped is created.

The model IDs are the same we used in the previous tutorial, when we changed our character's appearance to a cat. A list of all available models can be found here. `PedHash.Cat`, `PedHash.Deer`, `PedHash.AviSchwartzman`are all possible IDs we can assign to the NPC we want to create. We can create a new model variable, which we will name 'myPedModel' and assign it a model ID:

```
Model myPedModel = PedHash.AviSchwartzman;
```

The location where the NPC is created through a vector3 data type, which represents a vector in 3D space. This basically means a point that contains X, Y and Z coordinates. We can give absolute coordinates, making the Ped appear at a specific location in the game, but we can also use a location relative to our position in the game. In order not to risk making a Ped appear somewhere completely outside of our view – on some mountain or in the sea – let's look at a vector3 that points to a position in front of the player.

We want to establish the player with`Game.Player.Character`, followed by a function that retireve the player position within the game world. That's called by using `GetOffsetInWorldCoords`, which takes a vector3. The values of the X, Y and Z of the vector 3 offset the location based on the origin point represented by the player. Therefore, we can move the place where we want the Ped to appear by adding values to the X axis (left or right of player), Y axis (ahead or behind the player), and Z axis (above or below the player). To make a Ped appear in front of the player we can create a vector3 data type with 0 for X, 5 for Y and 0 for Z: `new Vector3(0, 5, 0)`. Let's make a vector3 variable, which we will name 'myPedSpawnPosition', assign it the values above for X, Y and Z coordinates from the player position.

```
Vector3 myPedSpawnPosition = Game.Player.Character.GetOffsetInWorldCoords(new Vector3(0
```

Now we can use the model and the position variables to spawn the NPC in front of the player. We'll create a Ped named 'myPed1' and use the `World.CreatePed` function with the two variables as parameters:

```
var myPed1 = World.CreatePed(myPedModel, myPedSpawnPosition);
```

Example code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GTA;
using GTA.Math;
using System.Windows.Forms;
using System.Drawing;
using GTA.Native;


namespace moddingTutorial
{
    public class moddingTutorial : Script
    {
        public moddingTutorial()
        {
            this.Tick += onTick;
            this.KeyUp += onKeyUp;
            this.KeyDown += onKeyDown;
        }

        private void onTick(object sender, EventArgs e) //this function gets executed continuousl
        {
        }

        private void onKeyUp(object sender, KeyEventArgs e)//everything inside here is executed o
        {
        }

        private void onKeyDown(object sender, KeyEventArgs e) //everything inside here is execute
        {
            //when pressing 'K'
            if(e.KeyCode == Keys.K)
            {
                //select a model and store it in a variable
                Model myPedModel = PedHash.AviSchwartzman;

        //create a position relative to the player
        Vector3 myPedSpawnPosition = Game.Player.Character.GetOffsetInWorldCoords(new Vector3(0,

        //create a Ped with the chosen model, spawning at the chosen position
        var myPed1 = World.CreatePed(myPedModel, myPedSpawnPosition);
```

```
            }
        }
    }
}
```

Control Multiple NPCs

You can create multiple NPCs and give them custom names. Let's create a human NPC and a cat NPC and call them Jim and MannyTheCat respectively:

```
var Jim = World.CreatePed(PedHash.AviSchwartzman, Game.Player.Character.GetOffsetInWor
var MannyTheCat = World.CreatePed(PedHash.Cat, Game.Player.Character.GetOffsetInWorldC
```

Try to kill one of the `Ped` NPCs you created by using the `Kill()`.

```
Jim.Kill();
```

Note that when you kill your `Ped` 'Jim', it falls on the floor and it won't actually respond to any call or task you will give it, but it's not removed from the game. To remove a specific `Ped` you have to use the `Delete`function, which will remove that instance (and will make the NPC disappear).

```
Jim.Delete();
```

To handle groups of NPCs we can use the `List` class. A `List` is a collection of objects, and a `List`of `Peds` allows us to store our NPCs. We can use an index to retrieve and control specific `Peds` in the group. You can see the reference for more detailed information.

Create a `List` of `Peds` named myPeds as a global variable in the public class `public class moddingTutorial : Script`.

```
List<Ped> myPeds = new List<Ped>();
```

In the onKeyDown function `private void onKeyDown(object sender, KeyEventArgs e)` create 5 new Peds with a For Loop

```
for (int i = 0; i < 5; i++)
{
    //spawn a new Ped called newPed
    var newPed = World.CreatePed(PedHash.Clown01SMY, Game.Player.Character.GetOffsetIn
    //add the new Ped to my list of Peds myPeds
    myPeds.Add(newPed);
}
```

Now all the 5 `Peds` are part of the myPeds[] `List`. You can control each Ped individually by calling their individual number ID in the group. The first spawn `Ped` is myPed[0], the last one is myPeds[4].

Tell the 1st spawned NPC to start wandering around:

```
myPeds[0].Task.WanderAround();
```

Kill the 2nd spawned NPC:

```
myPeds[1].Kill();
```

Tell the 3rd NPC to jump:

```
myPeds[2].Jump();
```

Tell the 4th NPC to walk toward the camera:

```
myPeds[3].GoTo(GameplayCamera.Position);
```

Tell the 5th NPC to put their hands up for 3 seconds:

```
myPeds[4].Task.HandsUp(3000);
```

Nearby NPCs

Script Hook V DOt Net provides a function `GetNearbyPeds`which groups all the `Peds` within a nearby radius from a character.

Create a new group that adds `Peds` which are closer than 20 meters from the player (add that as a global variable in `public class moddingTutorial : Script`):

```
Ped[] NearbyPeds = World.GetNearbyPeds(Game.Player.Character, 20f);
```

Use a Foreach Loop to get every `Ped` in the group and give them the task to put their hands up for a second:

```
foreach (Ped p in NearbyPeds)
{
    p.Task.HandsUp(1000);
}
```

`GetNearbyPeds` does not sort out individual `Peds`in the group based on distance, so we have to do a bit of manual filtering to get the nearest NPC within the chosen radius from the player character.

Define the global variables in the public class `public class moddingTutorial : Script`:

```
float lastDistance;
Ped nearestPed = null;
Ped oldNearestPed = null;
```

Get and parse the nearby NPCs in the OnTick function `private void onTick(object sender, EventArgs e)`:

```
//set radius
float maxDistance = 25f;
//get nearest peds
Ped[] pedsGroup = World.GetNearbyPeds(Game.Player.Character, maxDistance);

float lastDistance = maxDistance;
foreach (Ped ped in pedsGroup)
{
    float distance = ped.Position.DistanceTo(Game.Player.Character.Position);
    if (distance < lastDistance)
    {
        nearestPed = ped;
        lastDistance = distance;
    }
}

    if (nearestPed != null && oldNearestPed != nearestPed)
    {
        nearestPed.Task.HandsUp(1000);
    }
oldNearestPed = nearestPed;
```

Give Tasks to NPCs

A Ped can be given a task using the `Task` function, just like we did in the previous tutorial for the player character.

```
myPed1.Task.WanderAround();
```

Some tasks involve interacting with other characters (Peds or Game.Player.Character) or take different parameters like positions (vector3), duration (in milliseconds),

and other data types. We can give our NPC the task to fight against the player by using the `FightAgainst` function, which requires a Ped parameter – which in the case of the player is expressed as `Game.Player.Character`.

```
myPed1.Task.FightAgainst(Game.Player.Character); //give npc task to fight against player
```

Try to replace the task to "fight against" with "flee from (player)" , "hands up", "jump"... or some of the other available tasks.

See the TaskInvoker list for possible tasks, or click on the list of available tasks below.

List of Available Tasks

```
void AchieveHeading (float heading, int timeout=0)

void AimAt (Entity target, int duration)

void AimAt (Vector3 target, int duration)

void Arrest (Ped ped)

void ChatTo (Ped ped)

void Jump ()

void Climb ()

void ClimbLadder ()

void Cower (int duration)

void ChaseWithGroundVehicle (Ped target)

void ChaseWithHelicopter (Ped target, Vector3 offset)

void ChaseWithPlane (Ped target, Vector3 offset)

void CruiseWithVehicle (Vehicle vehicle, float speed, DrivingStyle style=DrivingStyle.Normal)

void DriveTo (Vehicle vehicle, Vector3 target, float radius, float speed, DrivingStyle style=Driv

void EnterAnyVehicle (VehicleSeat seat=VehicleSeat.Any, int timeout=-1, float speed=1f, EnterVehi

void EnterVehicle (Vehicle vehicle, VehicleSeat seat=VehicleSeat.Any, int timeout=-1, float speed
```

```
void FightAgainst (Ped target)

void FightAgainst (Ped target, int duration)

void FightAgainstHatedTargets (float radius)

void FightAgainstHatedTargets (float radius, int duration)

void FleeFrom (Ped ped, int duration=-1)

void FleeFrom (Vector3 position, int duration=-1)

void FollowPointRoute (params Vector3[] points)

void FollowPointRoute (float movementSpeed, params Vector3[] points)

void FollowToOffsetFromEntity (Entity target, Vector3 offset, float movementSpeed, int

void GoTo (Entity target, Vector3 offset=default(Vector3), int timeout=-1)

void GoTo (Vector3 position, int timeout=-1)

void GoStraightTo (Vector3 position, int timeout=-1, float targetHeading=0f, float dis

void GuardCurrentPosition ()

void HandsUp (int duration)

void LandPlane (Vector3 startPosition, Vector3 touchdownPosition, Vehicle plane=null)

void LeaveVehicle (LeaveVehicleFlags flags=LeaveVehicleFlags.None)

void LeaveVehicle (Vehicle vehicle, bool closeDoor)

void LeaveVehicle (Vehicle vehicle, LeaveVehicleFlags flags)

void LookAt (Entity target, int duration=-1)

void LookAt (Vector3 position, int duration=-1)

void ParachuteTo (Vector3 position)

void ParkVehicle (Vehicle vehicle, Vector3 position, float heading, float radius=20.0f

void PerformSequence (TaskSequence sequence)
```

```
void PlayAnimation (string animDict, string animName)

void PlayAnimation (string animDict, string animName, float speed, int duration, float playbackRa

void PlayAnimation (string animDict, string animName, float blendInSpeed, int duration, Animation

void PlayAnimation (string animDict, string animName, float blendInSpeed, float blendOutSpeed, in

void RappelFromHelicopter ()

void ReactAndFlee (Ped ped)

void ReloadWeapon ()

void RunTo (Vector3 position, bool ignorePaths=false, int timeout=-1)

void ShootAt (Ped target, int duration=-1, FiringPattern pattern=FiringPattern.Default)

void ShootAt (Vector3 position, int duration=-1, FiringPattern pattern=FiringPattern.Default)

void ShuffleToNextVehicleSeat (Vehicle vehicle=null)

void Skydive ()

void SlideTo (Vector3 position, float heading)

void StandStill (int duration)

void StartScenario (string name, float heading)

void StartScenario (string name, Vector3 position, float heading)

void SwapWeapon ()

void TurnTo (Entity target, int duration=-1)

void TurnTo (Vector3 position, int duration=-1)

void UseParachute ()

void UseMobilePhone ()

void UseMobilePhone (int duration)

void PutAwayParachute ()
```

```
void PutAwayMobilePhone ()

void VehicleChase (Ped target)

void VehicleShootAtPed (Ped target)

void Wait (int duration)

void WanderAround ()

void WanderAround (Vector3 position, float radius)

void WarpIntoVehicle (Vehicle vehicle, VehicleSeat seat)

void WarpOutOfVehicle (Vehicle vehicle)

void ClearAll ()

void ClearAllImmediately ()

void ClearLookAt ()

void ClearSecondary ()

void ClearAnimation (string animSet, string animName)
```

You can spawn a group of NPCs and give them individual tasks. You can also make them interact with each other (or with the player character). Here we spawn 3 NPCs and tell the to fight with each other.

```
//create a list of Peds
List<Ped> myPeds = new List<Ped>();

//create a list of Ped models
List<Model> myPedModel = new List<Model>();

//manually add models for each ped
myPedModel.Add(PedHash.Clown01SMY);
myPedModel.Add(PedHash.Doctor01SMM);
myPedModel.Add(PedHash.Abigail);


for(int i = 0; i < myPedModel.Count; i++)
{
```

```
    //spawn a new Ped for each model
    var newPed = World.CreatePed(myPedModel[i], Game.Player.Character.GetOffsetInWorldCoords(new
    //add the new Ped to my list of Peds
    myPeds.Add(newPed);
}

myPeds[0].Task.FightAgainst(myPeds[1]);
myPeds[1].Task.FightAgainst(myPeds[2]);
myPeds[2].Task.FightAgainst(myPeds[0]);
```

To clear a task at any given moment we can use the task `ClearAllImmediately();`. To stop our 3 NPCs from fighting each other we give them the task to stop everything they are doing immediately.

```
myPeds[0].Task.ClearAllImmediately();
myPeds[1].Task.ClearAllImmediately();
myPeds[2].Task.ClearAllImmediately();
```

Peace is restored in the universe. To remove the NPCs use `Delete()`.

```
myPeds[0].Delete();
myPeds[1].Delete();
myPeds[2].Delete();
```

Scenarios

Screnarios are short looping animations that can be triggered with the `Task` of type `StartScenario`. Unlike other animation types (see next chapter below), scenarios include props with the moevement of the character. Also, certain scenarios only work with specific model, so we won't be able to make a deer drink coffe or use a binocular (we'll make a deer pole dancing later don't worry).

Scenarios can be invoked by using the task `StartScenario(string name, Vector3 position, float heading)`. the `StartScenario` task function requires 3 paramenters: the name of the scenario we want to play, the position in which our Ped will be playing the scenario (through a Vector3 data type containing the XYZ coordinates), and the rotation degree (between 1 and 360° - note: this seems to work only when applied to the player character and not on Ped NPCs).

Try to play the scenario animation `WORLD_HUMAN_TOURIST_MAP` on our game character:

```
Game.Player.Character.Task.StartScenario("WORLD_HUMAN_TOURIST_MAP", Game.Player.Charact
```

To stop the scenario animation you can use `Game.Player.Character.Task.ClearAllImmediately();` like with every other task. You can find all available scenarios here or click on the list of below

List of available scenarios

```
WORLD_HUMAN_AA_COFFEE
WORLD_HUMAN_AA_SMOKE
WORLD_HUMAN_BINOCULARS
WORLD_HUMAN_BUM_FREEWAY
WORLD_HUMAN_BUM_SLUMPED
WORLD_HUMAN_BUM_STANDING
WORLD_HUMAN_BUM_WASH
WORLD_HUMAN_VALET
WORLD_HUMAN_CAR_PARK_ATTENDANT
WORLD_HUMAN_CHEERING
WORLD_HUMAN_CLIPBOARD
WORLD_HUMAN_CLIPBOARD_FACILITY
WORLD_HUMAN_CONST_DRILL
WORLD_HUMAN_COP_IDLES
WORLD_HUMAN_DRINKING
WORLD_HUMAN_DRINKING_FACILITY
WORLD_HUMAN_DRINKING_CASINO_TERRACE
WORLD_HUMAN_DRUG_DEALER
WORLD_HUMAN_DRUG_DEALER_HARD
WORLD_HUMAN_MOBILE_FILM_SHOCKING
WORLD_HUMAN_GARDENER_LEAF_BLOWER
WORLD_HUMAN_GARDENER_PLANT
WORLD_HUMAN_GOLF_PLAYER
WORLD_HUMAN_GUARD_PATROL
WORLD_HUMAN_GUARD_STAND
WORLD_HUMAN_GUARD_STAND_CASINO
WORLD_HUMAN_GUARD_STAND_CLUBHOUSE
WORLD_HUMAN_GUARD_STAND_FACILITY
WORLD_HUMAN_GUARD_STAND_ARMY
WORLD_HUMAN_HAMMERING
WORLD_HUMAN_HANG_OUT_STREET
WORLD_HUMAN_HANG_OUT_STREET_CLUBHOUSE
WORLD_HUMAN_HIKER
WORLD_HUMAN_HIKER_STANDING
WORLD_HUMAN_HUMAN_STATUE
WORLD_HUMAN_JANITOR
WORLD_HUMAN_JOG
```

```
WORLD_HUMAN_JOG_STANDING
WORLD_HUMAN_LEANING
WORLD_HUMAN_LEANING_CASINO_TERRACE
WORLD_HUMAN_MAID_CLEAN
WORLD_HUMAN_MUSCLE_FLEX
WORLD_HUMAN_MUSCLE_FREE_WEIGHTS
WORLD_HUMAN_MUSICIAN
WORLD_HUMAN_PAPARAZZI
WORLD_HUMAN_PARTYING
WORLD_HUMAN_PICNIC
WORLD_HUMAN_POWER_WALKER
WORLD_HUMAN_PROSTITUTE_HIGH_CLASS
WORLD_HUMAN_PROSTITUTE_LOW_CLASS
WORLD_HUMAN_PUSH_UPS
WORLD_HUMAN_SEAT_LEDGE
WORLD_HUMAN_SEAT_LEDGE_EATING
WORLD_HUMAN_SEAT_STEPS
WORLD_HUMAN_SEAT_WALL
WORLD_HUMAN_SEAT_WALL_EATING
WORLD_HUMAN_SEAT_WALL_TABLET
WORLD_HUMAN_SECURITY_SHINE_TORCH
WORLD_HUMAN_SIT_UPS
WORLD_HUMAN_SMOKING
WORLD_HUMAN_SMOKING_CLUBHOUSE
WORLD_HUMAN_SMOKING_POT
WORLD_HUMAN_SMOKING_POT_CLUBHOUSE
WORLD_HUMAN_STAND_FIRE
WORLD_HUMAN_STAND_FISHING
WORLD_HUMAN_STAND_IMPATIENT
WORLD_HUMAN_STAND_IMPATIENT_CLUBHOUSE
WORLD_HUMAN_STAND_IMPATIENT_FACILITY
WORLD_HUMAN_STAND_IMPATIENT_UPRIGHT
WORLD_HUMAN_STAND_IMPATIENT_UPRIGHT_FACILITY
WORLD_HUMAN_STAND_MOBILE
WORLD_HUMAN_STAND_MOBILE_CLUBHOUSE
WORLD_HUMAN_STAND_MOBILE_FACILITY
WORLD_HUMAN_STAND_MOBILE_UPRIGHT
WORLD_HUMAN_STAND_MOBILE_UPRIGHT_CLUBHOUSE
WORLD_HUMAN_STRIP_WATCH_STAND
WORLD_HUMAN_STUPOR
WORLD_HUMAN_STUPOR_CLUBHOUSE
WORLD_HUMAN_SUNBATHE
WORLD_HUMAN_SUNBATHE_BACK
WORLD_HUMAN_SUPERHERO
WORLD_HUMAN_SWIMMING
WORLD_HUMAN_TENNIS_PLAYER
```

```
WORLD_HUMAN_TOURIST_MAP
WORLD_HUMAN_TOURIST_MOBILE
WORLD_HUMAN_VEHICLE_MECHANIC
WORLD_HUMAN_WELDING
WORLD_HUMAN_WINDOW_SHOP_BROWSE
WORLD_HUMAN_YOGA
PROP_HUMAN_ATM
PROP_HUMAN_BBQ
PROP_HUMAN_BUM_BIN
PROP_HUMAN_BUM_SHOPPING_CART
PROP_HUMAN_MUSCLE_CHIN_UPS
PROP_HUMAN_MUSCLE_CHIN_UPS_ARMY
PROP_HUMAN_MUSCLE_CHIN_UPS_PRISON
PROP_HUMAN_PARKING_METER
PROP_HUMAN_SEAT_ARMCHAIR
PROP_HUMAN_SEAT_BAR
PROP_HUMAN_SEAT_BENCH
PROP_HUMAN_SEAT_BENCH_FACILITY
PROP_HUMAN_SEAT_BENCH_DRINK
PROP_HUMAN_SEAT_BENCH_DRINK_FACILITY
PROP_HUMAN_SEAT_BENCH_DRINK_BEER
PROP_HUMAN_SEAT_BENCH_FOOD
PROP_HUMAN_SEAT_BENCH_FOOD_FACILITY
PROP_HUMAN_SEAT_BUS_STOP_WAIT
PROP_HUMAN_SEAT_CHAIR
PROP_HUMAN_SEAT_CHAIR_DRINK
PROP_HUMAN_SEAT_CHAIR_DRINK_BEER
PROP_HUMAN_SEAT_CHAIR_FOOD
PROP_HUMAN_SEAT_CHAIR_UPRIGHT
PROP_HUMAN_SEAT_CHAIR_MP_PLAYER
PROP_HUMAN_SEAT_COMPUTER
PROP_HUMAN_SEAT_COMPUTER_LOW
PROP_HUMAN_SEAT_DECKCHAIR
PROP_HUMAN_SEAT_DECKCHAIR_DRINK
PROP_HUMAN_SEAT_MUSCLE_BENCH_PRESS
PROP_HUMAN_SEAT_MUSCLE_BENCH_PRESS_PRISON
PROP_HUMAN_SEAT_SEWING
PROP_HUMAN_SEAT_STRIP_WATCH
PROP_HUMAN_SEAT_SUNLOUNGER
PROP_HUMAN_STAND_IMPATIENT
CODE_HUMAN_CROSS_ROAD_WAIT
CODE_HUMAN_MEDIC_KNEEL
CODE_HUMAN_MEDIC_TEND_TO_DEAD
CODE_HUMAN_MEDIC_TIME_OF_DEATH
CODE_HUMAN_POLICE_CROWD_CONTROL
CODE_HUMAN_POLICE_INVESTIGATE
```

```
EAR_TO_TEXT
EAR_TO_TEXT_FAT
--- ---
WORLD_BOAR_GRAZING
WORLD_CAT_SLEEPING_GROUND
WORLD_CAT_SLEEPING_LEDGE
WORLD_COW_GRAZING
WORLD_COYOTE_HOWL
WORLD_COYOTE_REST
WORLD_COYOTE_WANDER
WORLD_COYOTE_WALK
WORLD_CHICKENHAWK_FEEDING
WORLD_CHICKENHAWK_STANDING
WORLD_CORMORANT_STANDING
WORLD_CROW_FEEDING
WORLD_CROW_STANDING
WORLD_DEER_GRAZING
WORLD_DOG_BARKING_ROTTWEILER
WORLD_DOG_BARKING_RETRIEVER
WORLD_DOG_BARKING_SHEPHERD
WORLD_DOG_SITTING_ROTTWEILER
WORLD_DOG_SITTING_RETRIEVER
WORLD_DOG_SITTING_SHEPHERD
WORLD_DOG_BARKING_SMALL
WORLD_DOG_SITTING_SMALL
WORLD_DOLPHIN_SWIM
WORLD_FISH_FLEE
WORLD_FISH_IDLE
WORLD_GULL_FEEDING
WORLD_GULL_STANDING
WORLD_HEN_FLEE
WORLD_HEN_PECKING
WORLD_HEN_STANDING
WORLD_MOUNTAIN_LION_REST
WORLD_MOUNTAIN_LION_WANDER
WORLD_ORCA_SWIM
WORLD_PIG_GRAZING
WORLD_PIGEON_FEEDING
WORLD_PIGEON_STANDING
WORLD_RABBIT_EATING
WORLD_RABBIT_FLEE
WORLD_RATS_EATING
WORLD_RATS_FLEEING
WORLD_SHARK_SWIM
WORLD_SHARK_HAMMERHEAD_SWIM
WORLD_STINGRAY_SWIM
```

We can give scenarios to multiple Peds just like we gave different tasks to NPCs in the section above. We can add a `List` to store multiple scenarios, just like we made a `List` to store multiple NPCs. Then we can control individual NPCs and trigger specific scenarios by going through the list items.

For example the following line will control the first NPC Ped we created, and made them play the third scenario in the list (note that items added to list start from 0, so the first one is actually number 0).

```
myPeds[0].Task.StartScenario(scenarios[2], myPeds[0].Position, 300f);
```

In the code below we press `G` to spawn 5 Peds of model Clown in front of the player character, and we add 5 scenarios to our list. Then press `H` to make each Ped play one of the scenarios. Use `J`to stop the task and `K` to delete the Peds and clear the list.

Example code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Drawing;
using GTA;
using GTA.Math;
using GTA.Native;

namespace moddingTutorial
{
    public class moddingTutorial : Script
    {
        List<string> scenarios = new List<string>();
        List<Ped> myPeds = new List<Ped>();

        public moddingTutorial()
        {
            this.Tick += onTick;
            this.KeyUp += onKeyUp;
            this.KeyDown += onKeyDown;
        }

        private void onTick(object sender, EventArgs e)
        {
```

```
        }

        private void onKeyUp(object sender, KeyEventArgs e)
        {

        }

        private void onKeyDown(object sender, KeyEventArgs e)
        {

            if (e.KeyCode == Keys.G)
            {
                //add scenarios to the list of scenarios
                scenarios.Add("WORLD_HUMAN_AA_COFFEE");
                    scenarios.Add("WORLD_HUMAN_TOURIST_MAP");
                    scenarios.Add("WORLD_HUMAN_TOURIST_MOBILE");
                    scenarios.Add("WORLD_HUMAN_BINOCULARS");
                    scenarios.Add("WORLD_HUMAN_PARTYING");
                    scenarios.Add("WORLD_HUMAN_MUSCLE_FLEX");

                //create 5 Clown NPCs and add the to the list of myPeds
                for (int i = 0; i < 5; i++)
                {
                    //spawn a new Ped called newPed
                    var newPed = World.CreatePed(PedHash.Clown01SMY, Game.Player.Character.GetOff
                    //add the new Ped to my list of Peds myPeds
                    myPeds.Add(newPed);
                }
            }
            if (e.KeyCode == Keys.H)
            {
                //give the scenarios to the Peds
                for (int i = 0; i < 5; i++)
                {
                        myPeds[i].Task.StartScenario(scenarios[i], myPeds[i].Position, 300f);
                }
            }
            if (e.KeyCode == Keys.J)
            {
                    //stop the task of each Ped
                    for (int i = 0; i < 5; i++)
                {
                        myPeds[i].Task.ClearAllImmediately();
                }
            }
```

```
            if (e.KeyCode == Keys.K)
            {
                //delete 5 Peds
                for (int i = 0; i < 5; i++)
                {
                    myPeds[i].Delete();
                }

                //and clear the list
                myPeds.Clear();
            }

        }
    }
}
```

Animations

We can get Peds to play specific animations from a larger database of different possible movements. To do this we can use the native function `TASK_PLAY_ANIM`. The function takes a lot of parameters (some of them still not exactly know), but here is the full function and a breakdown of each parameters.

```
Native.Function.Call(Native.Hash.TASK_PLAY_ANIM, thePed, sDict, sAnim, speed, speed *
```

`thePed` The Ped that will play the animation `sDict` The dictionary where the anim is located `sAnim` The anim name `speed` The play start speed (This is important to make smooth changes between anims) `speed * -1` Unknown `-1` Unknown `flags` Flags that you can set for the playback (see flags below) `0` Unknown `false` Unknown `bDisableLegIK` If the anim will ignore the leg/foot interaction with obstacles `false` Unknown

Flags for playback modes

```
```
normal = 0
    repeat = 1
    stop_last_frame = 2
    unk1 = 4
    unk2_air = 8
    upperbody = 16
    enablePlCtrl = 32
```
```

```
    unk3 = 64
    cancelable = 128
    unk4_creature = 256
    unk5_freezePos = 512
    unk6_rot90 = 1024
```

You need to request the animation dictionary before start using it in your script: `REQUEST_ANIM_DICT`. After that, wait for the animation to load (or you could check if it's loaded with the boolean `HAS_ANIM_DICT_LOADED`), before playing the animation.

Once you have requested your animation dictionary and it is loaded, you can play and stop the specific animation using `TASK_PLAY_ANIM` and `STOP_ANIM_TASK`.

```
//request animation dictionary
Function.Call(Hash.REQUEST_ANIM_DICT, "mini@strip_club@pole_dance@pole_a_2_stage");

//wait 100 ms to load the animation
Wait(100);

//play animation from animation dictionary using the player character
Function.Call(Hash.TASK_PLAY_ANIM, Game.Player.Character, "mini@strip_club@pole_dance@pole_a_2_st

//wait 5 secs
Wait(5000);

//stop the animation
Function.Call(Hash.STOP_ANIM_TASK, Game.Player.Character, "mini@strip_club@pole_dance@pole_a_2_st
```

Most information found for this functions were found here. More example code and information is avaiable there.

There are 6645 animation dictionaries and 35460 animation clips. You can see some of the possible animations in GTA V here. Here you can find a list of available dictionaries and animations.

Fun fact: deers seem to be able to do pole dance animations too.

Give animations to nearby peds.

```
//request animation dictionary
Function.Call(Hash.REQUEST_ANIM_DICT, "gestures@miss@fbi_5");

//wait for it to load
Wait(50);
```

```
//get nearby ped
Ped[] NearbyPeds = World.GetNearbyPeds(Game.Player.Character, 20f);

foreach (Ped p in NearbyPeds)
{
    //clear the peds of any tasks they might have
    p.Task.ClearAllImmediately();
    //play animation from animation dictionary
    Function.Call(Hash.TASK_PLAY_ANIM, p, "missfbi5ig_2", "crying_trevor", 8.0, 8.0 * -
}
```

Playing Sounds

We can control sounds from the game as well as playing external sound files. Here's an overview of a few different functions to can play speech recordings and audio from the GTA V audio bank, and a method to play a custom .wav file saved onside the "scripts" folder.

PLAY AMBIENT SPEECHES

The native function `_PLAY_AMBIENT_SPEECH1` and `_PLAY_AMBIENT_SPEECH2` both do the same thing: they make the character say a few lines, with the voice of the character model, and with their relative mouth and body animations. The speeches are, as the name suggest, ambient lines of dialogue that main characters and NPCs say throughout the game in different situations.

`_PLAY_AMBIENT_SPEECH1` will take as parameters the Ped that is going to speak, a string of the speech name, the speech parameter string and an integer that is set to 1.

Try to use the native function to make the player character say something from the `"GENERIC_BYE"` speech bank:

```
Function.Call(Hash._PLAY_AMBIENT_SPEECH1, Game.Player.Character, "GENERIC_BYE", "SPEECH
```

You will see that every time you use that function the character will say one possible variation ("bye", "later on", "bye now" if you use the character of Franklin) contained in the generic bye speech. If you change your character thespeech will change accordingly, although not every character has speech associated with them. For speech parameters, the easiest is to stick with "SPEECH_PARAMS_FORCE".

Here you can find a list of all available speech names.

List of speech names

```
// List of Speech Names found in gtav.ysc.decompiled (28 May 2015)

APOLOGY_NO_TROUBLE
BLOCKED_GENERIC
BUMP
CHAT_RESP
CHAT_STATE
COVER_ME
COVER_YOU
DODGE
DYING_HELP
DYING_MOAN
FALL_BACK
GENERIC_BYE
GENERIC_CURSE_HIGH
GENERIC_CURSE_MED
GENERIC_FRIGHTENED_HIGH
GENERIC_FRIGHTENED_MED
GENERIC_FUCK_YOU
GENERIC_HI
GENERIC_HOWS_IT_GOING
GENERIC_INSULT_MED
GENERIC_INSULT_HIGH
GENERIC_SHOCKED_HIGH
GENERIC_SHOCKED_MED
GENERIC_THANKS
GENERIC_WAR_CRY
HOOKER_CAR_INCORRECT
HOOKER_DECLINE_SERVICE
HOOKER_DECLINED
HOOKER_DECLINED_TREVOR
HOOKER_HAD_ENOUGH
HOOKER_LEAVES_ANGRY
HOOKER_OFFER_AGAIN
HOOKER_OFFER_SERVICE
HOOKER_REQUEST
HOOKER_SECLUDED
HOOKER_STORY_REVULSION_RESP
HOOKER_STORY_SARCASTIC_RESP
HOOKER_STORY_SYMPATHETIC_RESP
KIFFLOM_GREET
KILLED_ALL
PROVOKE_TRESPASS
PURCHASE_ONLINE
RELOADING
ROLLERCOASTER_CHAT_EXCITED
```

```
ROLLERCOASTER_CHAT_NORMAL
SEX_CLIMAX
SEX_FINISHED
SEX_GENERIC
SEX_GENERIC_FEM
SEX_ORAL
SEX_ORAL_FEM
SHOOT
SHOP_BANTER
SHOP_BANTER_FRANKLIN
SHOP_BANTER_TREVOR
SHOP_BROWSE
SHOP_BROWSE_ARMOUR
SHOP_BROWSE_BIG
SHOP_BROWSE_FRANKLIN
SHOP_BROWSE_GUN
SHOP_BROWSE_MELEE
SHOP_BROWSE_TATTOO_MENU
SHOP_BROWSE_THROWN
SHOP_BROWSE_TREVOR
SHOP_CUTTING_HAIR
SHOP_GIVE_FOR_FREE
SHOP_GOODBYE
SHOP_GREET
SHOP_GREET_FRANKLIN
SHOP_GREET_MICHAEL
SHOP_GREET_SPECIAL
SHOP_GREET_TREVOR
SHOP_GREET_UNUSUAL
SHOP_HAIR_WHAT_WANT
SHOP_NICE_VEHICLE
SHOP_NO_COPS
SHOP_NO_MESSING
SHOP_NO_WEAPON
SHOP_OUT_OF_STOCK
SHOP_REMOVE_VEHICLE
SHOP_SELL
SHOP_SELL_ARMOUR
SHOP_SELL_BRAKES
SHOP_SELL_BULLETPROOF_TYRES
SHOP_SELL_COSMETICS
SHOP_SELL_ENGINE_UPGRADE
SHOP_SELL_EXHAUST
SHOP_SELL_HORN
SHOP_SELL_REPAIR
SHOP_SELL_SUSPENSION
```

```
SHOP_SELL_TRANS_UPGRADE
SHOP_SELL_TURBO
SHOP_SHOOTING
SHOP_SPECIAL_DISCOUNT
SHOP_TATTOO_APPLIED
SHOP_TRY_ON_ITEM
SHOUT_THREATEN_GANG
SHOUT_THREATEN_PED
SOLICIT_FRANKLIN
SOLICIT_FRANKLIN_RETURN
SOLICIT_MICHAEL
SOLICIT_MICHAEL_RETURN
SOLICIT_TREVOR
SOLICIT_TREVOR_RETURN
STAY_DOWN
TAKE_COVER
```

PLAY AMBIENT SPEECHES WITH A DIFFERENT VOICE

We can also play an ambient speech with a different voice from the one of the character we are using. The function `_PLAY_AMBIENT_SPEECH_WITH_VOICE` takes as parameter the Ped that will speak, the speech ID, the speech voice model and speech parameters.

```
Function.Call(Hash._PLAY_AMBIENT_SPEECH_WITH_VOICE, Game.Player.Character, "GENERIC_BYE", "A_F_M
```

```
Function.Call(Hash._PLAY_AMBIENT_SPEECH_WITH_VOICE, Game.Player.Character, "GENERIC_BYE", "A_F_M_
```

Not every speech model has all the recorded ambient speech, so we must try and see what works and what doesn't.

PLAY PAIN SOUNDS

Apart from ambient speech, we can also play "pain" sounds. The native function `PLAY_PAIN` contains different kind of screams that can be played. Depending on the model, the screams will be more likely an integer of value 6, 7 or 8. Try to mess with the code find out what kind of screams can be played. Not for the faint of heart:

```
Function.Call(Hash.PLAY_PAIN, Game.Player.Character, 6, 0, 0);
```

You can disable and enable these sounds for each character using the native `DISABLE_PED_PAIN_AUDIO` and setting it to true or false:

```
Function.Call(Hash.DISABLE_PED_PAIN_AUDIO,Game.Player.Character , true);
```

PLAY SOUND FILES FROM GTA V SOUND BANK

We can also play a sound file from an audio bank of GTA V sound files. To do that we first load the audio bank, then we use the native function `PLAY_SOUND_FROM_ENTITY`, specify the audio file name, the character and the audio reference.

```
//load the audio bank
while (!Function.Call<bool>(Hash.REQUEST_SCRIPT_AUDIO_BANK, "Michael_2_Acid_Bath", 0,
//play the audio file
Function.Call(Hash.PLAY_SOUND_FROM_ENTITY, -1, "ACID_BATH_FALL", Game.Player.Character
```

PLAY EXTERNAL SOUND FILES

Finally we can play an external sound file by using the `SoundPlayer` object. `SoundPlayer` is simple but limited and only accepts .wav files. To use it, first add `using System.Media` at the very top of your code, to import the reference.

Create a new player as a global variable and load the file

```
SoundPlayer player = new SoundPlayer("./scripts/Eran_N.wav");
player.Load();
```

And on key press you can use play and stop to control playback

```
//play
player.Play();

//stop
player.Stop()
```

Teleporting

We can change the location of the player character or of any Ped or Vehicle entity by using the native function `SET_ENTITY_COORDS`. This function needs an entity and X, Y and Z coordinate to teleport to. We need to know the exact coordinates of the locations we want to teleport to, but thankfully the modding community forums provide lists with all available coordinates we can teleport to. Let's take the XYZ coordinates of the top of Mount Chiliad (the highest point in the game) to teleport our player character to.

```
LOCATION: Top of the Mt Chilad
COORDINATES: X:450.718 Y:5566.614 Z:806.183
```

To create a teleport function we will use a native function. Script Hook V Dot Net is a wrapper for the C++ ScriptHook, calling the functions in Scripthook to do things in the game. However, there are some functions that are not in Script Hook V Dot Net and in order to use these, we have to use the native calling from Script Hook.

Native functions are called with `Function.Call` followed by their corresponding hash name and parameters. They use this structure:

```
Function.Call(Hash.HASH_NAME, input_params);
```

The native function for teleporting expects the hash `SET_ENTITY_COORDS`, the `ped` entity to teleport, and the X, Y and Z coordinates to teleport the character to. `Function.Call(Hash.SET_ENTITY_COORDS, Ped ped, X, Y, Z, 0, 0, 1);`

The function to teleport the player character to the top of Moutn Chiliad is:

```
//Teleport to the top of Mount Chiliad
Function.Call(Hash.SET_ENTITY_COORDS, Game.Player.Character, 450.718f, 5566.614f, 806.183f, 0, 0,
```

See this list of locations to find their respective coordinates or click on the list below

List of Locations with Coordinates

```
INDOOR LOCATIONS

Strip Club DJ Booth X:126.135 Y:-1278.583 Z:29.270

Blaine County Savings Bank X:-109.299 Y:6464.035 Z:31.627

Police Station X:436.491 Y: -982.172 Z:30.699

Humane Labs Entrance X:3619.749 Y:2742.740 Z:28.690

Burnt FIB Building X:160.868 Y:-745.831 Z:250.063

10 Car Garage Back Room X:223.193 Y:-967.322 Z:99.000

Humane Labs Tunnel X:3525.495 Y:3705.301 Z:20.992

Ammunation Office X:12.494 Y:-1110.130 Z: 29.797

Ammunation Gun Range X: 22.153 Y:-1072.854 Z:29.797
```

```
Trevor's Meth Lab X:1391.773 Y:3608.716 Z:38.942

Pacific Standard Bank Vault X:255.851 Y: 217.030 Z:101.683

Lester's House X:1273.898 Y:-1719.304 Z:54.771

Floyd's Apartment X:-1150.703 Y:-1520.713 Z:10.633

FIB Top Floor X:135.733 Y:-749.216 Z:258.152

IAA Office X:117.220 Y:-620.938 Z:206.047

Pacific Standard Bank X:235.046 Y:216.434 Z:106.287

Fort Zancudo ATC entrance X:-2344.373 Y:3267.498 Z:32.811

Fort Zancudo ATC top floor X:-2358.132 Y:3249.754 Z:101.451

Torture Room X: 147.170 Y:-2201.804 Z:4.688

OUTDOOR LOCATIONS

Main LS Customs X:-365.425 Y:-131.809 Z:37.873

Very High Up X:-129.964 Y:8130.873 Z:6705.307

IAA Roof X:134.085 Y:-637.859 Z:262.851

FIB Roof X:150.126 Y:-754.591 Z:262.865

Maze Bank Roof X:-75.015 Y:-818.215 Z:326.176

Top of the Mt Chilad X:450.718 Y:5566.614 Z:806.183

Most Northerly Point X:24.775 Y:7644.102 Z:19.055

Vinewood Bowl Stage X:686.245 Y:577.950 Z:130.461

Sisyphus Theater Stage X:205.316 Y:1167.378 Z:227.005

Galileo Observatory Roof X:-438.804 Y:1076.097 Z:352.411

Kortz Center X:-2243.810 Y:264.048 Z:174.615

Chumash Historic Family Pier X:-3426.683 Y:967.738 Z:8.347
```

Paleto Bay Pier X:-275.522 Y:6635.835 Z:7.425

God's thumb X:-1006.402 Y:6272.383 Z:1.503

Calafia Train Bridge X:-517.869 Y:4425.284 Z:89.795

Altruist Cult Camp X:-1170.841 Y:4926.646 Z:224.295

Maze Bank Arena Roof X:-324.300 Y:-1968.545 Z:67.002

Marlowe Vineyards X:-1868.971 Y:2095.674 Z:139.115

Hippy Camp X:2476.712 Y:3789.645 Z:41.226

Devin Weston's House X:-2639.872 Y:1866.812 Z:160.135

Abandon Mine X:-595.342 Y: 2086.008 Z:131.412

Weed Farm X:2208.777 Y:5578.235 Z:53.735

Stab City X: 126.975 Y:3714.419 Z:46.827

Airplane Graveyard Airplane Tail X:2395.096 Y:3049.616 Z:60.053

Satellite Dish Antenna X:2034.988 Y:2953.105 Z:74.602

Satellite Dishes X: 2062.123 Y:2942.055 Z:47.431

Windmill Top X:2026.677 Y:1842.684 Z:133.313

Sandy Shores Building Site Crane X:1051.209 Y:2280.452 Z:89.727

Rebel Radio X:736.153 Y:2583.143 Z:79.634

Quarry X:2954.196 Y:2783.410 Z:41.004

Palmer-Taylor Power Station Chimney X: 2732.931 Y: 1577.540 Z:83.671

Merryweather Dock X: 486.417 Y:-3339.692 Z:6.070

Cargo Ship X:899.678 Y:-2882.191 Z:19.013

Del Perro Pier X:-1850.127 Y:-1231.751 Z:13.017

Play Boy Mansion X:-1475.234 Y:167.088Z:55.841

```
Jolene Cranley-Evans Ghost X:3059.620 Y:5564.246 Z:197.091

NOOSE Headquarters X:2535.243 Y:-383.799 Z:92.993

Snowman X: 971.245 Y:-1620.993 Z:30.111

Oriental Theater X:293.089 Y:180.466 Z:104.301

Beach Skatepark X:-1374.881 Y:-1398.835 Z:6.141

Underpass Skatepark X:718.341 Y:-1218.714 Z: 26.014

Casino X:925.329 Y:46.152 Z:80.908

University of San Andreas X:-1696.866 Y:142.747 Z:64.372

La Puerta Freeway Bridge X: -543.932 Y:-2225.543 Z:122.366

Land Act Dam X: 1660.369 Y:-12.013 Z:170.020

Mount Gordo X: 2877.633 Y:5911.078 Z:369.624

Little Seoul X:-889.655 Y:-853.499 Z:20.566

Epsilon Building X:-695.025 Y:82.955 Z:55.855 Z:55.855

The Richman Hotel X:-1330.911 Y:340.871 Z:64.078

Vinewood sign X:711.362 Y:1198.134 Z:348.526

Los Santos Golf Club X:-1336.715 Y:59.051 Z:55.246

Chicken X:-31.010 Y:6316.830 Z:40.083

Little Portola X:-635.463 Y:-242.402 Z:38.175

Pacific Bluffs Country Club X:-3022.222 Y:39.968 Z:13.611

Vinewood Cemetery X:-1659993 Y:-128.399 Z:59.954

Paleto Forest Sawmill Chimney X:-549.467 Y:5308.221 Z:114.146

Mirror Park X:1070.206 Y:-711.958 Z:58.483

Rocket X:1608.698 Y:6438.096 Z:37.637
```

```
El Gordo Lighthouse X:3430.155 Y:5174.196 Z:41.280
```

Content Replication Assignment

Teleport the player to a beach, spawn ten whales on the shore and generate an NPC wandering aroud them and take a screenshot in the style of HAPP V2.

Chapter 7

Hyperrealism

//intro to the artistic current of photorealism and hyperrealism in painting and drawing, connected to the idea of photorealism and simulation in games, relation between game and cinema and the attempt to simulate life itself, not just photographic image. relationship between the game and the physical world, the way virtual spaces influence and shape society (training self driving cars in GTA V, CGI shaping architecture of buildings and object…), the blurrying of the lines between virtual and physical…

8k by Aram Bartholl

Aram Bartholl, *8k*, installation view

Aram Bartholl, *8k*, installation view

More about 8k

Getting there

- Land Act Dam, Tataviam Mountains

Readings

Tutorial

Using Props to Light a Scene

In addition to NPCs, we can also spawn and control objects. There are more than 15000 objects in the GTA V database that can be accessed and search through this index. From yachts to paintings, from flower vases to rocks, they vary in scale and properties and constitute all the objects that are present in the game world.

Objects are referred as `Prop` variables in the code, and similarly to `Ped` and `Vehicle` they are a `GTA Entity` data type. To create a new prop we use the `World.CreateProp` function. This needs the following parameters: a string with the prop file name, the relative of absolute 3D coordinates of where the prop should be created, the 3D rotation value (optional, if not specified the prop will have default rotation values), a true/false boolean to set the object as dynamic or static, a true/false boolean for snapping the object on the ground or spawning it at an arbitrary position.

```
GTA.World.CreateProp(model : Model , position: Vector3, rotation:
Vector3, dynamic : bool, onGround : bool)
```

Let's create a `Prop` variable called MyProp and let's make an alarm siren appearing 3 meters in front of the player, with default rotation, dynamic and snapping to its original position:

```
Prop MyProp = World.CreateProp("xm_prop_x17_sub_alarm_lamp", Game.Player.Character.GetOffsetInWor
```

By default the siren is placed on top of a ceiling, so it will appear above the player.

To delete a prop we can call the `Prop` variable we created and use the `Delete()` function:

```
MyProp.Delete();
```

We can also remove Collisions by using the `IsCollisionEnabled`boolean to false:

```
MyProp.IsCollisionEnabled(false);
```

There are several hundreds different light objects within the game's database. We can use these object to manipulate the light for our shots, just like a photographer would do in the studio.

Try spawning “prop_spot_clamp_02” to get harsh spot lights

```
Prop MyProp = World.CreateProp("prop_spot_clamp_02", Game.Player.Character.GetOffsetIn
```

//To do: introduction to lighting in photography //Hard, soft, specular and diffuse light //3 Point Lighting Technique

Below is a list of objects casting lights which can be used to light up the scene in different ways.

Lighting props

```
 //STUDIO WHITE TRIPOD LIGHT (COLD)
Prop MyProp = World.CreateProp("ch_prop_tunnel_tripod_lampa", Game.Player.Character.Ge
```

```
//STUDIO WHITE TRIPOD LIGHT (WARMER)
Prop MyProp = World.CreateProp("xm_prop_base_tripod_lampb", Game.Player.Character.GetO
```

```
//STUDIO WHITE TRIPOD LIGHT (STRONGER)
Prop MyProp = World.CreateProp("xm_prop_base_tower_lampa", Game.Player.Character.GetOf
```

```
//LIGHT BOX (WARM)
Prop MyProp = World.CreateProp("xm_prop_base_wall_lampa", Game.Player.Character.GetOffs
```

```
//LIGHT BOX (COLD)
Prop MyProp = World.CreateProp("xm_prop_base_wall_lampb", Game.Player.Character.GetOffs
```

```
//WHITE NEON LIGHT (no tripod)
Prop MyProp = World.CreateProp("xm_prop_base_tripod_lampc", Game.Player.Character.GetO
```

```
//NEON LIGHT GROUP OF 3 (vertical)
Prop MyProp = World.CreateProp("xm_prop_lab_tube_lampa_group3", Game.Player.Character.
```

```
//NEON LIGHT GROUP OF 6 (vertical, different colours: modify ending with _g, _p, _r, _
Prop MyProp = World.CreateProp("xm_prop_lab_tube_lampa_group6_g", Game.Player.Characte
```

```
//ARENA LIGHTS
Prop MyProp = World.CreateProp("xs_propintarena_lamps_01a", Game.Player.Character.GetO
```

```
//STROBO LIGHTS
Prop MyProp = World.CreateProp("ba_prop_battle_lights_fx_riga", Game.Player.Character.
```

```
//4 FLOATING SPOTS LIGHTBOX (modify ending to get different color hues: _lr1 to _lr9)
Prop MyProp = World.CreateProp("ba_prop_battle_lights_int_03_lr1", Game.Player.Character.GetOffse

//WHITE WASH CEILING LAMP
Prop MyProp = World.CreateProp("prop_chall_lamp_02", Game.Player.Character.GetOffsetInWorldCoords

//WHITE SPOT LAMP
Prop MyProp = World.CreateProp("prop_spot_clamp_02", Game.Player.Character.GetOffsetInWorldCoords

//MULTIPLE WHITE LIGHTS SETUP
Prop MyProp = World.CreateProp("v_ilev_carmodlamps", Game.Player.Character.GetOffsetInWorldCoords

//SOFT WHITE WASH
Prop MyProp = World.CreateProp("v_ilev_fh_lampa_on", Game.Player.Character.GetOffsetInWorldCoords

//CEILING WHITE NEON (HARSH)
Prop MyProp = World.CreateProp("xm_base_cia_lamp_ceiling_01", Game.Player.Character.GetOffsetInWo

//CEILING WHITE SPOT (HARSH)
Prop MyProp = World.CreateProp("xm_prop_base_silo_lamp_01a", Game.Player.Character.GetOffsetInWor

//FLOOR SOFT WHITE WASH TILE
Prop MyProp = World.CreateProp("xm_base_cia_lamp_floor_01a", Game.Player.Character.GetOffsetInWor

//RED SIREN CEILING LIGHT (SIREN ON = xm_prop_x17_sub_al_lamp_on SIREN OFF = xm_prop_x17_sub_al_l
Prop MyProp = World.CreateProp("xm_prop_x17_sub_alarm_lamp", Game.Player.Character.GetOffsetInWor

//RED WASH
Prop MyProp = World.CreateProp("v_ilev_cd_lampal", Game.Player.Character.GetOffsetInWorldCoords(r

//LIGHT BLUE WASH CEILING LAMP
Prop MyProp = World.CreateProp("prop_chall_lamp_01n", Game.Player.Character.GetOffsetInWorldCoord
```

Setting the Time of the Day

We can control the time and the light for our shots to the second in GTA V. We can specify the exact time of the day with the native function `SET_CLOCK_TIME`, followed by hour, minute and second. Note that the sun sets at 5:30 a.m. and goes down at 8 p.m.

Set the imte of the day to 15:45:

```
Function.Call(Hash.SET_CLOCK_TIME, 15, 45, 00);
```

Controlling the Weather

We can choose among the following weather options using the native function `SET_WEATHER_TYPE_NOW_PERSIST`:

"CLEAR" "EXTRASUNNY" "CLOUDS" "OVERCAST" "RAIN" "CLEARING" "THUNDER" "SMOG" "FOGGY" "XMAS" "SNOWLIGHT" "BLIZZARD"

Set the weather to blizzard:

```
Function.Call(Hash.SET_WEATHER_TYPE_NOW_PERSIST, "BLIZZARD");
```

Scripting Cinematic Fade Out / In

Script Hook has native functions to create a fead to/from black. The function hash are `DO_SCREEN_FADE_OUT` and `DO_SCREEN_FADE_IN` and they are followed by the number of milliseconds to go from full black to showing the scene and viceversa.

Let's create a fade to black over 3 seconds when we press the letter key 'O':

```
if (e.KeyCode == Keys.O)
{
    Function.Call(Hash.DO_SCREEN_FADE_OUT, 3000);
}
```

And a fade over 3 seconds when we press the letter key 'I':

```
if (e.KeyCode == Keys.I)
{
    Function.Call(Hash.DO_SCREEN_FADE_IN, 3000);
}
```

We could also create an automated check in our onTick loop, which keeps seeing if the screen has been faded to black. We can use the native function `IS_SCREEN_FADED_OUT` which is a boolean data type. This means it will return either true or false. If it returns true, it means the screen has been faded out.

Let's add an "if" statement in our onTick loop to chek if the screen has been faded to black, and if so we teleport somewhere else and we call a fade in over 3 second:

```
if (Function.Call<bool>(Hash.IS_SCREEN_FADED_OUT))
{
    Function.Call(Hash.SET_ENTITY_COORDS, Game.Player.Character, -1374.881f, -1398.835f, 6.141f,
    Wait(500);
    GTA.Native.Function.Call(Hash.DO_SCREEN_FADE_IN, 3000);
}
```

Now if you try to hit the 'O' key, the screen will fade out, and then it will automatically fade in again.

Natural Vision Evolved Mod

Natural Vision Evolved (NVE) is a graphic mod develped by Jamal Rashid, aka Razed. This mod enhances GTA V's lighting, weather effects, ambient colours, world textures, building models, pushing the photo-realism and cinematic looks. While the mod contains settings for different hardware settings, it's recommended to have a relatively powerful PC with a good graphic card. Here you can find minimum and recommended requirements for Natural Vision Evolved mod.

Installation and setup:

- Go to razedmods.com/gta-v and download Natural Vision Evolved (6.2 Gb).
- Go to openiv.com/ and download Open IV, Open 'ovisetup' and install Open IV on your computer.
- Select GTA V Windows. Choose Grand Theft Auto V folder `C:\Program Files (x86)\Steam\steamapps\common\Grand Theft Auto V`
- Once Open IV is open, go to your file window select the `Tools` menu on top of the window, and select `ASI Manager`. In `ASI Manager` install all options: ASI Loader, OpenIV.ASI and openCamera.
- Select `Tools` again and click `Options`. Click on the `"mods" folder` tab and select `Allow edit mode only for archive inside "mods" folder`. Click `Close`.
- Select `Edit mode` at the top right of the window. Select `OK` on the pop up window.
- Now you can add your mod to the mods folder.
- Open the NVE mod folder, extract and select NaturalVision Installer PART ONE and drag it to Open IV. Install the file and select "mod folder" when asked to choose. After that is complete, extract and select

NaturalVision Installer PART TWO and drag it to Open IV. Install the file and select "mod folder" when asked to choose. The Installers have to be executed in order: first do PART ONE, and then do PART TWO.

- Go back to your downloads and inside the NVE folder you can choose some optional addons. Install/Uninstall them with Open IV as above. Always choose to select "mod" folder and select install.
- Open GTA V, press `ESC` to bring up the menu and go to `SETTINGS`. Adjust the graphics quality, making sure `Shader Quality`, `Particle Quality` and `Post FX`are set to `<Very High>`. Restart the game to make change in effect. More details about installation are provided in the README file inside the NVE zip folder.

ReShade

ReShade is a generic post-processing injector for games and video software developed by crosire.

Installation and setup:

- Go to reshade.me and download the latest version
- Open ReShade Setup and select Browse and navigate to your Grand Theft Auto V folder and select GTA 5.exe. Leave Direct X 10/11/12 checked and click Next. When asked to select presets to install click on Skip.
- Select SweetFX by CeeJay.dk and all the effect packages you want to install (OtisFX by Otis_Inf and CobraFX by SirCobra are particularly interesting for simulation of analogue photography, including depth of field) and complete installation. You can also select and install all filters, so you can play around with them all.
- Run GTA V and press the `Home key` to bring up the menu. The first time you bring up Reshade, it will offer to give a short tutorial on how to choose and create presets. You can use the ReShade Default preset to turn on and off all the different filters. Each filter can be adjusted by modifying its parameters at the bottom of the panel.
- Create a new preset by duplicating the ReShade Default preset and giving it a custom name. Mess around with its setting to see how it affects the visuals of the game. Thanks to different effects and presets, the aesthetics of the game world can be configured to obtain looks that are completely different from the original graphics created by the deleopers and designers of the game.

Content Replication Assignment

Chapter 8

Glitch Art

//intro to error in photography (Clément Chéroux) and glitch art. photographying the glitch as the process of creating photorealism, exploring the materials of the game world, investigating the creation of textures, of physics,

Captures by Raphael Brunk

Raphael Brunk, *Capture75011.12_23* , 2016

Raphael Brunk, *Capture55326.4_19*, 2016

More about Captures

Getting there

- 3668 Wild Oats Dr, Vinewood Hills

Readings

Tutorial

Controlling the Game Camera

We can detach the camera from the player's character and move freely around the game world. We instantiate a new `Camera` variable called `FreeCam`. We define it as the new world camera with the same position and rotation of the gameplay camera:

```
FreeCam = World.CreateCamera(GameplayCamera.Position, GameplayCamera.Rotation, Gameplay
```

We then define our new camera as the main camera with `World.RenderingCamera = FreeCam`. In order to control the camera we basically its vectors in 3D coordinates and multiply with a number over a certain direction to move towards a specific destination. A free camera – unlike the main character view – also allows us to move through the game architecture and terrain, revealing the construction of the game world. Press `O` to toggle the free camera On and off.

Example code

```
//this code was kindly provided by LeeC22 on https://gtaforums.com/topic/981454-free-c

using System;
using System.Windows.Forms;
using GTA;
using GTA.Math;

using Control = GTA.Control;

namespace BasicFreeCamTest
{
    public class cBasicFreeCamTest : Script
    {
        private Keys ActivationKey = Keys.O;
        private bool FreeCamActive = false;
        private Camera FreeCam;

        public cBasicFreeCamTest()
        {
            Tick += onTick;
            KeyUp += onKeyUp;
            Aborted += onAborted;

            Interval = 0;
        }

        private void onTick(object sender, EventArgs e)
        {
            // Exits from the loop if the game is loading
            if (Game.IsLoading) return;

            if (FreeCamActive) UpdateFreeCam();
        }
```

```
        private void UpdateFreeCam()
        {
            float deltaTime = Game.LastFrameTime;
            float speed = 5f;
            float camSpeed = speed * deltaTime;
            float rotSpeed = 40f;
            float camRotSpeed = rotSpeed * deltaTime;

            Game.DisableAllControlsThisFrame(2);

            Vector3 camForward = FreeCam.Direction.Normalized; //FreeCam.ForwardVector
            Vector3 camRight = Vector3.Cross(Vector3.WorldUp, camForward); //FreeCam.RightVector
            Vector3 camUp = Vector3.Cross(camRight, camForward); //FreeCam.UpVector?

                if (Game.IsDisabledControlJustPressed(2, Control.FrontendCancel))
            {
                World.RenderingCamera = null;
                FreeCam.Destroy();
                FreeCamActive = false;
                return;
            }

            float fbSpeedMult = Game.GetDisabledControlNormal(2, Control.MoveUpDown);
            float lrSpeedMult = Game.GetDisabledControlNormal(2, Control.MoveLeftRight);
            float yawSpeedMult = Game.GetDisabledControlNormal(2, Control.LookLeftRight);
            float pitchSpeedMult = Game.GetDisabledControlNormal(2, Control.LookUpDown);

            float fbSpeed = camSpeed * fbSpeedMult;
            float lrSpeed = camSpeed * lrSpeedMult;
            float yawSpeed = camRotSpeed * yawSpeedMult;
            float pitchSpeed = camRotSpeed * pitchSpeedMult;

            Vector3 camLR = camRight * -lrSpeed;
            Vector3 camFB = camForward * -fbSpeed;
            Vector3 camMove = camLR + camFB;

            Vector3 camRot = new Vector3(pitchSpeed, 0, -yawSpeed);

            FreeCam.Position += camMove;
            FreeCam.Rotation += camRot;
        }

        private void onKeyUp(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == ActivationKey)
```

```
            {
                //toggle the freecam on/off
                FreeCamActive = !FreeCamActive;

                if (FreeCamActive) //ON
                {
                    FreeCam = World.CreateCamera(GameplayCamera.Position, GameplayCamer
                    World.RenderingCamera = FreeCam;
                    UI.ShowSubtitle("FreeCam ON", 1000);
                }
                else //OFF
                {
                    World.RenderingCamera = null;
                    FreeCam.Destroy();
                    UI.ShowSubtitle("FreeCam OFF", 1000);
                }
            }
        }

        private void onAborted(object sender, EventArgs e)
        {
            World.RenderingCamera = null;
        }
    }
}
```

Recording and Replaying Camera Movements

We can record the movement of the camera by storing its posiiton and movement every frame in a list of Vector3 variables. Then we can simply access the list and assign the position of the camera at each frame. This is useful to create reusable camera movements, like tracking shots and cinematic camera movements. Once we have achieved the desired camera movement, we can spawn objects in the scene and adjust color grading before saving the screengrab. In this example, turn on the free camera by pressing `O` and then start and stop the recording of the camera movement by pressing `I`. To toggle the replay camera press `K`.

Example code

```
//this code was extended by code for controlling the camera by LeeC22 on https://gtafo
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;

using System.Windows.Forms;
using System.Drawing;
using GTA;
using GTA.Math;
using GTA.Native;


using Control = GTA.Control;

namespace BasicFreeCamTest
{
    public class cBasicFreeCamTest : Script
    {
        private Keys ActivationKey = Keys.O;
        private Keys RecordKey = Keys.I;
        private Keys ReplayKey = Keys.K;
        private bool FreeCamActive = false;
        private bool RecCamActive = false;
        private int CamFramesIndex = 0;
        private bool ReplayCamActive = false;
        private Camera FreeCam;
        private Camera ReplayCam;
        List<Vector3> RecCamPos = new List<Vector3>();
        List<Vector3> RecCamRot = new List<Vector3>();

        public cBasicFreeCamTest()
        {
            Tick += onTick;
            KeyUp += onKeyUp;
            Aborted += onAborted;

            Interval = 0;
        }

        private void onTick(object sender, EventArgs e)
        {
            // Exits from the loop if the game is loading
            if (Game.IsLoading) return;

            if (FreeCamActive) UpdateFreeCam();

            if (RecCamActive) UpdateRecCam();
```

```
            if (ReplayCamActive) UpdateReplayCam();
        }

        private void UpdateRecCam()
        {
            RecCamPos.Add(FreeCam.Position);
            RecCamRot.Add(FreeCam.Rotation);
            UI.ShowSubtitle("Recording Camera ON", 100);
        }
        private void UpdateReplayCam()
        {

            Game.DisableAllControlsThisFrame(2);

            ReplayCam.Position = RecCamPos[CamFramesIndex];
            ReplayCam.Rotation = RecCamRot[CamFramesIndex];

            CamFramesIndex++;
            if (CamFramesIndex >= RecCamPos.Count)
            {
                CamFramesIndex = RecCamPos.Count - 1;
                ReplayCamActive = false;
            }
            else
            {
                UI.ShowSubtitle("Playing Back Camera\nPlaying Back Frame #" + CamFrames
            }
        }

        private void UpdateFreeCam()
        {
            float deltaTime = Game.LastFrameTime;
            float speed = 2f;
            float camSpeed = speed * deltaTime;
            float rotSpeed = 80f;
            float camRotSpeed = rotSpeed * deltaTime;

            Game.DisableAllControlsThisFrame(2);

            Vector3 camForward = FreeCam.Direction.Normalized;
            Vector3 camRight = Vector3.Cross(Vector3.WorldUp, camForward);
            Vector3 camUp = Vector3.Cross(camRight, camForward);

            if (Game.IsDisabledControlJustPressed(2, Control.FrontendCancel))
            {
```

```
            World.RenderingCamera = null;
            FreeCam.Destroy();
            FreeCamActive = false;
            return;
        }

        float fbSpeedMult = Game.GetDisabledControlNormal(2, Control.MoveUpDown);
        float lrSpeedMult = Game.GetDisabledControlNormal(2, Control.MoveLeftRight);
        float yawSpeedMult = Game.GetDisabledControlNormal(2, Control.LookLeftRight);
        float pitchSpeedMult = Game.GetDisabledControlNormal(2, Control.LookUpDown);

        float fbSpeed = camSpeed * fbSpeedMult;
        float lrSpeed = camSpeed * lrSpeedMult;
        float yawSpeed = camRotSpeed * yawSpeedMult;
        float pitchSpeed = camRotSpeed * pitchSpeedMult;

        Vector3 camLR = camRight * -lrSpeed;
        Vector3 camFB = camForward * -fbSpeed;
        Vector3 camMove = camLR + camFB;

        Vector3 camRot = new Vector3(pitchSpeed, 0, -yawSpeed);

        FreeCam.Position += camMove;
        FreeCam.Rotation += camRot;
    }

    private void onKeyUp(object sender, KeyEventArgs e)
    {
        if (e.KeyCode == ActivationKey)
        {
            //toggle the freecam on/off
            FreeCamActive = !FreeCamActive;

            if (FreeCamActive) //ON
            {
                FreeCam = World.CreateCamera(GameplayCamera.Position, GameplayCamera.Rotation
                World.RenderingCamera = FreeCam;
                UI.ShowSubtitle("Free Camera ON", 1000);
            }
            else //OFF
            {
                World.RenderingCamera = null;
                FreeCam.Destroy();
                UI.ShowSubtitle("Free Camera OFF", 1000);
            }
```

```
            }
            if (e.KeyCode == RecordKey)
            {
                RecCamActive = !RecCamActive;
                if(RecCamActive)
                {
                    //Clear the array of positions and rotations
                    RecCamPos.Clear();
                    RecCamRot.Clear();
                    //reset index
                    CamFramesIndex= 0;
                    //disable replay cam
                    ReplayCamActive= false;
                    //activate free cam if it's off
                    if (!FreeCamActive)
                    {
                        FreeCam = World.CreateCamera(GameplayCamera.Position, GameplayC
                        World.RenderingCamera = FreeCam;
                        FreeCamActive = true;

                    }
                }
                else
                {
                    UI.ShowSubtitle("Recording Camera OFF\nRecorded Frames = " + RecCam
                }
            }
            if (e.KeyCode == ReplayKey)
            {
                ReplayCamActive = !ReplayCamActive;

                if(ReplayCamActive)
                {
                    CamFramesIndex = 0;
                    ReplayCam = World.CreateCamera(RecCamPos[0], RecCamRot[0], Gameplay
                    RecCamActive = false;
                    FreeCamActive = false;
                }
                else
                {
                    RecCamActive = false;
                    CamFramesIndex = 0;
                }
            }
        }
```

```
        private void onAborted(object sender, EventArgs e)
        {
            World.RenderingCamera = null;
        }
    }
}
```

Attaching a Camera to an Entity

We can also attach a camera to a car, a prop or an NPC. The Camera object in `GTA` has `AttachTo` function we can call, followed by two paramenters: the entity we want to attach the camera to and the position relative to the entity. Attaching a camera to an NPC can provide interesting views and perspectives. Here we create a cat NPC by pressing `G` and attach a camera to it. We can switch the camera view by pressing `SHIFT + K` and let the cat follow the player by pressing `H`.

Example code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GTA;
using GTA.Math;
using System.Windows.Forms;
using System.Drawing;
using GTA.Native;
using System.IO;


namespace moddingTutorial
{
    public class moddingTutorial : Script
    {
        Vector3 myCamPos;
        int CamSelect = 0;
        Ped newPed = null;
        Camera myCam;

        public moddingTutorial()
```

```
        {
            this.Tick += onTick;
            this.KeyUp += onKeyUp;
            this.KeyDown += onKeyDown;
        }

        private void onTick(object sender, EventArgs e) //this function gets executed
        {
            //exits from the loop if the game is loading
            if (Game.IsLoading) return;

            //update the cameras if the ped is spawn
            if (newPed != null)
            {
                //create the cameras if none have been created yet.
                if (myCam == null)
                {
                    UI.ShowSubtitle("Set new camera");
                    myCam = World.CreateCamera(Vector3.Zero, newPed.Rotation, 50f);
                    // Set the camera position (relative pos)
                    myCamPos = new Vector3(0, 0, 1f);
                }
                //attach the cameras
                myCam.AttachTo(newPed, myCamPos);
                //sync rotation
                myCam.Rotation = newPed.Rotation;
            }
        }


        private void onKeyUp(object sender, KeyEventArgs e)//everything inside here is
        {
            //press control+K to switch between gameplay default camera and the NPC ca
            if (e.KeyCode == Keys.K && e.Modifiers == Keys.Shift && newPed != null)
            {
                CamSelect = (CamSelect + 1) % 2;
                switch (CamSelect)
                {
                    case 0: World.RenderingCamera = null;
                        UI.ShowSubtitle("Showing Gameplay Cam View");
                        break;
                    case 1: World.RenderingCamera = myCam;
                        UI.ShowSubtitle("Showing NPC Cam View");
                        break;
                }
```

```
            }
        }

        private void onKeyDown(object sender, KeyEventArgs e) //everything inside here is execute
        {
            if(e.KeyCode == Keys.G)
            {
                //spawn new Ped
                newPed = World.CreatePed(PedHash.Cat, Game.Player.Character.GetOffsetInWorldCoord
            }

            if (e.KeyCode == Keys.H)
            {
                //follow player (persistent)
                Function.Call(Hash.TASK_FOLLOW_TO_OFFSET_OF_ENTITY, newPed.Handle, Game.Player.Ch
                //look at player
                newPed.Task.LookAt(Game.Player.Character);
            }
            if (e.KeyCode == Keys.J)
            {
                //stop NPC
                newPed.Task.ClearAll();
            }
            if (e.KeyCode == Keys.L)
            {
                //delete ped
                newPed.Delete();
            }
        }
    }
}
```

Creating and Switching between Multiple Cameras

The easiest way to create multiple cameras is position invisible Ped NPCs in specific locations, make them invisible with the native function `SET_ENTITY_VISIBLE` and attach a camera to them. We can switch between the different cameras with `SHIFT + K`. These can be used as different cameras to surveil and look at scenes in different locations within the game world.

Example code

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using GTA;
using GTA.Math;
using System.Windows.Forms;
using System.Drawing;
using GTA.Native;

namespace ModdingTutorial
{
    public class ModdingTutorial : Script
    {
        Vector3 myCamPos;
        int CamSelect = 0;
        int CamCount = 3;

        List<Ped> myPeds = new List<Ped>();
        Model myModel = PedHash.Abigail;
        List<Vector3> myLocs = new List<Vector3>();
        Camera newCam= null;
        List<Camera> myCam = new List<Camera>();


        public ModdingTutorial()
        {
            this.Tick += onTick;
            this.KeyUp += onKeyUp;
            this.KeyDown += onKeyDown;

            myLocs.Add(new Vector3(450.178f, 5566.614f, 806.183f)); //Mt.Chiliad
            myLocs.Add(new Vector3(24.775f, 7644.102f, 18.055f)); //Most Northern Poin
            myLocs.Add(new Vector3(150.126f, -754.591f, 261.865f)); //FIB Roof

            for (int i = 0; i < CamCount; i++)
            {
                //create Ped
                var newPed = World.CreatePed(myModel, myLocs[i]);
                Function.Call(Hash.SET_ENTITY_VISIBLE, newPed, false, 0);
                myPeds.Add(newPed);

                //create Cam
                newCam = World.CreateCamera(Vector3.Zero, myPeds[i].Rotation, 50f);
                myCam.Add(newCam);
                myCamPos = new Vector3(0, 0, 1f);
```

```
            }

        }

        private void onTick(object sender, EventArgs e)
        {
            //update Cams
            for (int i = 0; i < CamCount; i++)
            {
                myCam[i].AttachTo(myPeds[i], myCamPos);
                myCam[i].Rotation = myPeds[i].Rotation;
            }
        }

        private void onKeyUp(object sender, KeyEventArgs e)
        {
            //switch between cameras when pressing SHIFT + K
            if(e.KeyCode == Keys.K && e.Modifiers == Keys.Shift && myPeds[2] != null)
            {
                CamSelect = (CamSelect + 1) % 4;
                switch(CamSelect)
                {
                    case 0: World.RenderingCamera = null;
                        UI.ShowSubtitle("Showing Gameplay Cam View");
                        break;
                    case 1: World.RenderingCamera = myCam[0];
                        UI.ShowSubtitle("Showing Cam 1");
                        break;
                    case 2: World.RenderingCamera = myCam[1];
                        UI.ShowSubtitle("Showing Cam 2");
                        break;
                    case 3: World.RenderingCamera = myCam[2];
                        UI.ShowSubtitle("Showing Cam 3");
                        break;
                }
            }
        }

        private void onKeyDown(object sender, KeyEventArgs e)
        {
        }
    }
}
```

Switching Character through Satellite Camera View

We can use the native function `START_PLAYER_SWITCH` to enable the satellite view animation from the game, and get transported to a different location where an NPC or player avatar is spawned. Press `G` to move across divverent NPCs at different location through the satellite camera transition.

Example code

```
/*
    this was adapted from code shared by LeeC22 on gtaforums.com
    https://gtaforums.com/topic/951002-c-looking-for-player-switch-sample-solved-by-me/
*/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GTA;
using GTA.Math;
using System.Windows.Forms;
using System.Drawing;
using GTA.Native;
using System.IO;


namespace moddingTutorial
{
    public class moddingTutorial : Script
    {
        Ped newPed = null;
        Vector3 SwitchLocation2;
        List<Vector3> switchLocations = new List<Vector3>();
        int index = 0;
        List<String> models = new List<String>();
        int modelIndex = 0;

        public moddingTutorial()
        {
            this.Tick += onTick;
            this.KeyUp += onKeyUp;
            this.KeyDown += onKeyDown;

        //add locations to the switchLocations list
```

```
        switchLocations.Add(new Vector3(24.775f, 7644.102f, 18.055f)); //Most Northerly Point
        switchLocations.Add(new Vector3(-595.342f, 2086.008f, 130.412f)); //Mine
        switchLocations.Add(new Vector3(150.126f, -754.591f, 261.865f)); //FIB Roof

    //add models to the models list
        models.Add("s_m_m_doctor_01");
        models.Add("s_m_m_migrant_01");
        models.Add("a_c_cormorant");
        models.Add("a_c_deer");
        models.Add("a_c_pug");
    }

    private void onTick(object sender, EventArgs e) //this function gets executed continuousl
    {

        //If the character switch is in process
        if (Function.Call<bool>(Hash.IS_PLAYER_SWITCH_IN_PROGRESS))
        {
            //If Switch State is 8 - that's the point when it starts dropping to the player
            if (Function.Call<int>(Hash.GET_PLAYER_SWITCH_STATE) == 8)
            {
                //Set the player to the switch location
                Game.Player.Character.Position = switchLocations[index];

                //Generate the hash for the chosen model
                int poshHash = Game.GenerateHash(models[modelIndex]);

                //Create the model
                Model poshModel = new Model(poshHash);

                //Check if it is valid
                if (poshModel.IsValid)
                {
                    //Wait for it to load, should be okay because it was used to create the t
                    while (!poshModel.IsLoaded)
                    {
                        Wait(100);
                    }

                    //Change the player model to the target ped model
                    Function.Call(Hash.SET_PLAYER_MODEL, Game.Player, poshHash);

                    //Let the game clean up the created Model
                    poshModel.MarkAsNoLongerNeeded();
                }
```

```
                else
                {
                    //Falls to here if the model valid check fails
                    Function.Call(Hash.SET_PLAYER_MODEL, Game.Player, (int)PedHash
                }

                //Delete the target ped as it's no longer needed
                newPed.Delete();


                // Set the switch outro based on the gameplay camera position
                // Function.Call((Hash)0xC208B673CE446B61, camPos.X, camPos.Y, cam

                Function.Call((Hash)0xC208B673CE446B61, GameplayCamera.Position.X,

                //Call this unknown native that seems to finish things off
                Function.Call(Hash._0x74DE2E8739086740);

            //Make the character wander around autonomously
                Game.Player.Character.Task.WanderAround();
            }
        }

    }


    private void onKeyUp(object sender, KeyEventArgs e)
    {

    }

    private void onKeyDown(object sender, KeyEventArgs e)
    {
        if (e.KeyCode == Keys.G)
        {
    //Stop previous tasks
            Game.Player.Character.Task.ClearAll();

            //Move the index to the next location
            index++;
            if (index >= switchLocations.Count) index = 0;

    //Move the index to the new ped model
            modelIndex++;
            if (modelIndex >= models.Count) modelIndex = 0;
```

```
                //Create the ped to switch to
                newPed = World.CreatePed(models[modelIndex], switchLocations[index]);

                //Native function to initiate the switch Function.Call(Hash.START_PLAYER_SWITCH,
                Function.Call(Hash.START_PLAYER_SWITCH, Game.Player.Character.Handle, newPed.Hand

            }
        }
    }
}
```

Menyoo Mod Trainer

Game trainers are a kind of game modification that changes its behavior using addresses and values. Trainers are used to gain unfair advantage in games and cheating, but they are also used by players to "train" themselves under different game conditions. In GTA V a trainer mod is very useful for creative practices, as it allows players to modify many aspects of the game that extend the possibilities offered by the official Scene Director and Rockstar Editor. They do not require scripting and offer control of the game world through a more intuitive graphic interface.

Menyoo is one of many available mod trainers for GTA V. It's the most popular trainer because of its incredibly wide range of features, controlling NPCs, vehicles, props and scenes, and allowing players to generate custom interiors and objects from GTA V's database of entities, dynamic scenarios, and entire sets and scenes for machinima and photographic projects.

Installation and setup

Menyoo requires Scripthook V and Scripthook V Dot Net which we already installed in order to enable our scripts (check chapter 5 if for some reasons you do not have them installed).

- In order to install the Menyoo Single Player Trainer go to github.com/MAFINS/MenyooSP/releases and download the `MenyooSP.zip` file in the Assets section of the page. Open its content and select the `menyooStuff` folder and `Menyoo.asi` file. Copy and paste these in your GTA V directory.

- Run GTA V and press `F8` to bring up the Menyoo Trainer menu. From there you can control many of the functions we are learning to script, including character models, animations, and teleporting.

//https://forums.gta5-mods.com/topic/64/menyoo-object-spooner-tutorial

Scripting will always give you more accurate control of what you can achieve, but a trainer mod is very useful in showing what's possible and it's a much more accessible tool for those who are less incline to code. Finally, all of the tools in this guide are not mutually exclusive, but can be combined together: menyoo can be used with custom scripts, ReShade, screenshotting, Scene Director and Rockstar Editor, allowing the player to really have complete control of the game and its world. In the example below we have used code to spawn 5 clown NPCs and to position the game camera, the weather manipulated in Menyoo to create a meteor shower, and we used a shallow depth of field lens and color grading in ReShade.

CodeWalker

CodeWalker is an application created by dexyfex that renders an interactive 3D Map for GTAV. Through CodeWalker we are able to modify the architecture and objects inside the map, replacing textures and images with custom content.

Installation and setup

- Go to the CodeWalker GTA V 3D Map + Editor page and click download. Extract the content of the .zip folder and open the CodeWalker application. Once the GTAV installation folder has been found, the world view will load by default. Use `WASD Keys` to move, and `Mouse Drag` to rotate the camera view. `Mouse Wheel` zooms in/out and controls the movement speed.

- Click on the arrow on the top left of the screen, then select `Enable Mods` and `Enable DLC`.

- On top of the panel click on `Options > Lighting`. Deselect `Deferred shading` and in `Time of Day` set the time to 12:00. Then click on `Save Settings`.

Creating Custom Maps

- Next we are going to hit the `T` key and select the `Select objects` tool from the tool bar.

- Click on the `Move` tool, which allows to select things by right clicking on them. Hovering on the object will highlight the bounding box of the model, right clicking it will enable the movement of the object. Try dragging an object in 3D space, clicking and dragging one of the 3D axis arrows.

- Undo the changes by pressing `Undo` from the tool bar or `Ctrl + Z` until the object is back in its original position.
- Now press the first tool on the left in the tool bar: `New`. This will open the Project window.
- Select an object with the `Select objects` tool and you will see that the object information is updated in the Project window, including the object's position, rotation and file name (archetype). Choose an object and click `Add to Project`. This will add the object to our ymap.

Note: do not delete assets from the game map. Simply select an object that can be added to the project.

- With the `Select objects` tool and `Move` tool selected, hold `SHIFT` and drag the object to multiply the object. In this example we have multiplied a chair a few times, moved them and rotated them.
- Once you are happy with your changes, go to `File > New > Ymap File` in the Project window. This creates a new ymap called `map1.ymap`.
- We can also add objects that are not in the immediate vicinity of the location we want to add them to. Select map1.ymap in the Project window, and from the window menu press `Ymap > New Entity`. This will create a default egg. In the Project window go to Archetype and replace the file name with the one of the object you want. You can refer to this database to find the file name reference of each object easily (or use the `Select objects` tools and copy the Archetype info from your desired object).
- Once you are done click on the `Save` icon and save your file in a folder titled YMAP.
- In the Project window now go to `Tools` and click on `Manifest Generator...`. Click on `Save _manifest.yml` and save them as _manifest1.yml //memo: actually this is only needed for custom YMap on FiveM, so we could skip it here
- Go to Open IV > mods > update > x64 > dlcpacks, click Edit Mode and create a folder named custom_maps. Inside the folder copy the file dlc.rpf that you can find here.
- Now go to Open IV > mods > update > update.rpf > common > data. Find dlclist.xml, richt click on it and choose edit. Scroll to the end of the file and add `<Item>dlcpacks:/custom_maps/</Item> above </Paths>` and save.
- Finally go to Open IV > mods > update > x64 > dlcpacks > custom_maps > dlc.rpf > x64 > levels > gta5 > _citye > maps > Custom_maps.rpf. Make sure Edit Mode is on and place your .ymap file here.

- Now open GTA V and go to the location where you made the changes and see if they are there.
- Depending on the kind of objects you have place, and the flags you set, you will be able to interact with them, and the game will also recognize them and span some NPCs on them.

Editing and replacing textures

Another way in which we can intervene in the game world and modify it, is to replace existing images with custom ones. Every object in the game is made of a 3D model and one or more texture files attached to it. Textures are are 2D image files that are applied to 3D surfaces. In this example we'll swap the existing image of a billboard banner with a custom image.

For editing textures in game, we need two additional pieces of software installed before we go ahead: Adobe Photoshop and NVIDIA Texture Tools Exporter Plugin for Adobe Photoshop). Note that versions prior to Photoshop CC (5.0 - CS6) require the legacy version of the plugin.

- Open CodeWalker and find a texture you want to replace. Use the `Select objects` to move objects around until you find where the texture you want to change is. We are going to replace a billboard banner with a custom image. More specifically we'll replace the banner containing the "Fame or Shame" illustration with the posters of the exhibition How to Win at Photography. In this case we can see that the billboard structure is not part of the image, and that the banner is actually part of the ground model.
- In the CodeWalker menu on the right go to `Selection` and check the filename of the object that contains our image. In our case that is "sp1_04_ground". This is the name of the .ydr file that contains the map model.
- Open Open IV and copy "sp1_04_ground" into Open IV's search bar on the top of the window. Then click on `Search in all` under the "No items match your search" text. Select and double click on "sp1_04_ground.ydr" and click on `Copy to "mods" folder`. This will create a duplicate of the files so that we can make changes to them without corrupting the original files.
- Double click on "sp1_04_ground.ydr" to bring up OpenIV Model Viewer and inspect the 3D model. Click on `[+] Add texture` from the menu on the bottom right and select the different .ytd texture files associated with the model. If you select "spi_04.ytd" or "spi_04+hi.ytd" you'll load the texture of the billboard banner. These .ytd files are textures that are applied to the 3D model, and the two files are the ones that are

loaded by the game when the camera is closer and further from the object respectively.

- Close the Model Viewer and back in Open IV look for the "spi_04.ytd" texture file. Double click on it to bring up Open IV Texture Editor. Inside the .ydt file you will see all the .dds files contained in it. We have 2 files in "spi_04.ytd". We can preview them by selecting them and we can find the banner image in "sp1_04_rss_hc_bbrd_01". We select that, click `Export selected` and save the file somewhere on our computer.
- We open the file we saved in Photoshop. Modify the image as you want and then flatten all layers (ctrl+E) in Photoshop. Then choose `Save as a copy` and select .DDS type file and replace the file. The NVIDIA DDS Plugin interface will pop up. Here it's important to set the file type tp the same kind as the original .dds file. In this example the file is a DXT1 type, which is equivalent to the "BC1a 4bpp with 1 bit alpha" option. Select that and hit `Save`
- Close Photoshop and back in Open IV go back to "spi_04.ytd" and bring up Open IV Texture Editor. Select "sp1_04_rss_hc_bbrd_01" again and now click `Replace`. Select the modified file form your computer and click `Save`.
- Do the same for "spi_04+hi.ytd". Try to make the image as similar as possible so that when the game switches between the files (according to the player distance) the change is seamless.
- When all the textures are modified with our custom image, relaunch CodeWalker and verify that the image has changed.
- Finally launch GTA 5 and you should be able to find your texture appearing inside the game.

//try modifying the postcards at the Hookies family seafood restaurant, located on the Great Ocean Highway in North Chumash. -> check the prop_venice_racks.ytd -> prop_postcard.dds (part of the prop_postcard_rack.ydr).

//delete objects (note: if Max LOD > LOD shows the object, you should not delete it, just move it but dont delete it)

//generate and save manifestNAME.ymf // copy ymf and ymap files to server

Content Replication Assignment

Chapter 9

Meta-Photography

// General intro on the artistic reflection on the photographic medium itself, the tradition of conceptual photography and the connection to the simulation of the camera, photography about phtoography

GTA V Photography Bot

Crossroad of Realities by Benoit Paillé

Readings

Tutorial

Virtual Keys

The most reliable way to send keyboard and mouse input is the SendInput function in user32.dll.

The SendInput function takes three parameters: the number of inputs, an array of INPUT for the inputs we want to send, and the size of our INPUT struct. The INPUT struct includes an integer that indicates the type of input and a union for the inputs that will be passed.

Keys are mapped to Direct Input Keyboard hex codes. You can find a list of all hex codes for each key here or below.

List of DirectX key mappings

```
Value   Macro         Symbol
```

```
------------------------
0x01    DIK_ESCAPE  Esc
0x02    DIK_1         1
0x03    DIK_2         2
0x04    DIK_3         3
0x05    DIK_4         4
0x06    DIK_5         5
0x07    DIK_6         6
0x08    DIK_7         7
0x09    DIK_8         8
0x0A    DIK_9         9
0x0B    DIK_0         0
0x0C    DIK_MINUS     -
0x0D    DIK_EQUALS  =
0x0E    DIK_BACK      Back Space
0x0F    DIK_TAB     Tab
0x10    DIK_Q         Q
0x11    DIK_W         W
0x12    DIK_E         E
0x13    DIK_R         R
0x14    DIK_T         T
0x15    DIK_Y         Y
0x16    DIK_U         U
0x17    DIK_I         I
0x18    DIK_O         O
0x19    DIK_P         P
0x1A    DIK_LBRACKET  [
0x1B    DIK_RBRACKET  ]
0x1C    DIK_RETURN  Enter
0x1D    DIK_LContol Ctrl (Left)
0x1E    DIK_A         A
0x1F    DIK_S         S
0x20    DIK_D         D
0x21    DIK_F         F
0x22    DIK_G         G
0x23    DIK_H         H
0x24    DIK_J         J
0x25    DIK_K         K
0x26    DIK_L         L
0x27    DIK_SEMICOLON   ;
0x28    DIK_APOSTROPHE  '
0x29    DIK_GRAVE     `
0x2A    DIK_LSHIFT  Shift (Left)
0x2B    DIK_BACKSLASH   \
0x2C    DIK_Z         Z
0x2D    DIK_X         X
```

```
0x2E    DIK_C         C
0x2F    DIK_V         V
0x30    DIK_B         B
0x31    DIK_N         N
0x32    DIK_M         M
0x33    DIK_COMMA     ,
0x34    DIK_PERIOD  .
0x35    DIK_SLASH     /
0x36    DIK_RSHIFT  Shift (Right)
0x37    DIK_MULTIPLY    * (Numpad)
0x38    DIK_LMENU     Alt (Left)
0x39    DIK_SPACE     Space
0x3A    DIK_CAPITAL Caps Lock
0x3B    DIK_F1      F1
0x3C    DIK_F2      F2
0x3D    DIK_F3      F3
0x3E    DIK_F4      F4
0x3F    DIK_F5      F5
0x40    DIK_F6      F6
0x41    DIK_F7      F7
0x42    DIK_F8      F8
0x43    DIK_F9      F9
0x44    DIK_F10     F10
0x45    DIK_NUMLOCK Num Lock
0x46    DIK_SCROLL  Scroll Lock
0x47    DIK_NUMPAD7 7 (Numpad)
0x48    DIK_NUMPAD8 8 (Numpad)
0x49    DIK_NUMPAD9 9 (Numpad)
0x4A    DIK_SUBTRACT  - (Numpad)
0x4B    DIK_NUMPAD4 4 (Numpad)
0x4C    DIK_NUMPAD5 5 (Numpad)
0x4D    DIK_NUMPAD6 6 (Numpad)
0x4E    DIK_ADD     + (Numpad)
0x4F    DIK_NUMPAD1 1 (Numpad)
0x50    DIK_NUMPAD2 2 (Numpad)
0x51    DIK_NUMPAD3 3 (Numpad)
0x52    DIK_NUMPAD0 0 (Numpad)
0x53    DIK_DECIMAL . (Numpad)
0x57    DIK_F11     F11
0x58    DIK_F12     F12
0x64    DIK_F13     F13 (NEC PC-98)
0x65    DIK_F14     F14 (NEC PC-98)
0x66    DIK_F15     F15 (NEC PC-98)
0x70    DIK_KANA      Kana  Japenese Keyboard
0x79    DIK_CONVERT Convert Japenese Keyboard
0x7B    DIK_NOCONVERT   No Convert  Japenese Keyboard
```

```
0x7D    DIK_YEN     ¥     Japenese Keyboard
0x8D    DIK_NUMPADEQUALS    =   NEC PC-98
0x90    DIK_CIRCUMFLEX  ^   Japenese Keyboard
0x91    DIK_AT      @   NEC PC-98
0x92    DIK_COLON     : NEC PC-98
0x93    DIK_UNDERLINE   _   NEC PC-98
0x94    DIK_KANJI     Kanji Japenese Keyboard
0x95    DIK_STOP      Stop  NEC PC-98
0x96    DIK_AX      (Japan AX)
0x97    DIK_UNLABELED   (J3100)
0x9C    DIK_NUMPADENTER Enter (Numpad)
0x9D    DIK_RCONTROL    Ctrl (Right)
0xB3    DIK_NUMPADCOMMA , (Numpad)  NEC PC-98
0xB5    DIK_DIVIDE    / (Numpad)
0xB7    DIK_SYSRQ       Sys Rq
0xB8    DIK_RMENU       Alt (Right)
0xC5    DIK_PAUSE       Pause
0xC7    DIK_HOME        Home
0xC8    DIK_UP        ↑ (Arrow up)
0xC9    DIK_PRIOR       Page Up
0xCB    DIK_LEFT        ←   (Arrow left)
0xCD    DIK_RIGHT       →   (Arrow right)
0xCF    DIK_END       End
0xD0    DIK_DOWN        ↓   (Arrow down)
0xD1    DIK_NEXT        Page Down
0xD2    DIK_INSERT    Insert
0xD3    DIK_DELETE    Delete
0xDB    DIK_LWIN        Windows
0xDC    DIK_RWIN        Windows
0xDD    DIK_APPS        Menu
0xDE    DIK_POWER       Power
0xDF    DIK_SLEEP       Windows
```

Example code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GTA;
using GTA.Math;
using System.Windows.Forms;
using System.Drawing;
using GTA.Native;
```

```
using System.IO;
using System.Runtime.InteropServices;

namespace moddingTutorial
{
    public class moddingTutorial : Script
    {

        //this bunch of stuff is to control keyboard and mouse
        [StructLayout(LayoutKind.Sequential)]
        public struct KeyboardInput
        {
            public ushort wVk;
            public ushort wScan;
            public uint dwFlags;
            public uint time;
            public IntPtr dwExtraInfo;
        }
        [StructLayout(LayoutKind.Sequential)]
        public struct MouseInput
        {
            public int dx;
            public int dy;
            public uint mouseData;
            public uint dwFlags;
            public uint time;
            public IntPtr dwExtraInfo;
        }
        [StructLayout(LayoutKind.Sequential)]
        public struct HardwareInput
        {
            public uint uMsg;
            public ushort wParamL;
            public ushort wParamH;
        }
        [StructLayout(LayoutKind.Explicit)]
        public struct InputUnion
        {
            [FieldOffset(0)] public MouseInput mi;
            [FieldOffset(0)] public KeyboardInput ki;
            [FieldOffset(0)] public HardwareInput hi;
        }
        public struct Input
        {
            public int type;
```

```
        public InputUnion u;
    }
    [Flags]
    public enum InputType
    {
        Mouse = 0,
        Keyboard = 1,
        Hardware = 2
    }
    [Flags]
    public enum KeyEventF
    {
        KeyDown = 0x0000,
        ExtendedKey = 0x0001,
        KeyUp = 0x0002,
        Unicode = 0x0004,
        Scancode = 0x0008
    }
    [Flags]
    public enum MouseEventF
    {
        Absolute = 0x8000,
        HWheel = 0x01000,
        Move = 0x0001,
        MoveNoCoalesce = 0x2000,
        LeftDown = 0x0002,
        LeftUp = 0x0004,
        RightDown = 0x0008,
        RightUp = 0x0010,
        MiddleDown = 0x0020,
        MiddleUp = 0x0040,
        VirtualDesk = 0x4000,
        Wheel = 0x0800,
        XDown = 0x0080,
        XUp = 0x0100
    }

    [DllImport("user32.dll", SetLastError = true)]
    private static extern uint SendInput(uint nInputs, Input[] pInputs, int cbSize);
    [DllImport("user32.dll")]
    private static extern IntPtr GetMessageExtraInfo();
    bool pressKey = false;


    public moddingTutorial()
```

```
        {
            this.Tick += onTick;
            this.KeyUp += onKeyUp;
            this.KeyDown += onKeyDown;
        }

        private void onTick(object sender, EventArgs e) //this function gets executed
        {
        }

        private void onKeyUp(object sender, KeyEventArgs e)
        {
        }

        private void onKeyDown(object sender, KeyEventArgs e)
        {

            if (e.KeyCode == Keys.H)
            {
                //define the pressing of the arrow up key
                Input[] keyUP_press = new Input[]
                {
                  new Input
                  {
                    type = (int)InputType.Keyboard,
                    u = new InputUnion
                    {
                      ki = new KeyboardInput
                      {
                        wVk = 0,
                        wScan = 0xC8, // key ARROW UP
                        dwFlags = (uint)(KeyEventF.KeyDown | KeyEventF.Scancode), //ke
                        dwExtraInfo = GetMessageExtraInfo()
                      }
                    }
                  }
                };

                //define the release of the arrow up key
                Input[] keyUP_release = new Input[]
                {
                  new Input
                  {
                    type = (int)InputType.Keyboard,
                    u = new InputUnion
```

```
                    {
                      ki = new KeyboardInput
                      {
                        wVk = 0,
                        wScan = 0xC8, //key ARROW UP
                        dwFlags = (uint)(KeyEventF.KeyUp | KeyEventF.Scancode), //key release
                        dwExtraInfo = GetMessageExtraInfo()
                      }
                    }
                  }
                };

                //we need to send a key press and add a delay before releasing it otherwise it's
                //send key press of arrow up key
                SendInput((uint)keyUP_press.Length, keyUP_press, Marshal.SizeOf(typeof(Input)));
                //delay half a second
                Wait(500);
                //send key release of arrow up key
                SendInput((uint)keyUP_release.Length, keyUP_release, Marshal.SizeOf(typeof(Input)
            }
        }
    }
}
```

GTA V Photographer Bot Script

```
//#define debug


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GTA;
using GTA.Math;
using System.Windows.Forms;
using System.Drawing;
using GTA.Native;
using System.IO;
using System.Runtime.InteropServices;
```

```
namespace moddingTutorial
{
    public class moddingTutorial : Script
    {

        bool botOn = false; //turn the bot ON/OFF
        List<Vector3> switchLocations = new List<Vector3>(); //list of locations to te
        int index;

        //KEYBOARD AND MOUSE INPUT STUFF
        [StructLayout(LayoutKind.Sequential)]
        public struct KeyboardInput
        {
            public ushort wVk;
            public ushort wScan;
            public uint dwFlags;
            public uint time;
            public IntPtr dwExtraInfo;
        }
        [StructLayout(LayoutKind.Sequential)]
        public struct MouseInput
        {
            public int dx;
            public int dy;
            public uint mouseData;
            public uint dwFlags;
            public uint time;
            public IntPtr dwExtraInfo;
        }
        [StructLayout(LayoutKind.Sequential)]
        public struct HardwareInput
        {
            public uint uMsg;
            public ushort wParamL;
            public ushort wParamH;
        }
        [StructLayout(LayoutKind.Explicit)]
        public struct InputUnion
        {
            [FieldOffset(0)] public MouseInput mi;
            [FieldOffset(0)] public KeyboardInput ki;
            [FieldOffset(0)] public HardwareInput hi;
        }
        public struct Input
        {
```

```
        public int type;
        public InputUnion u;
    }
    [Flags]
    public enum InputType
    {
        Mouse = 0,
        Keyboard = 1,
        Hardware = 2
    }
    [Flags]
    public enum KeyEventF
    {
        KeyDown = 0x0000,
        ExtendedKey = 0x0001,
        KeyUp = 0x0002,
        Unicode = 0x0004,
        Scancode = 0x0008
    }
    [Flags]
    public enum MouseEventF
    {
        Absolute = 0x8000,
        HWheel = 0x01000,
        Move = 0x0001,
        MoveNoCoalesce = 0x2000,
        LeftDown = 0x0002,
        LeftUp = 0x0004,
        RightDown = 0x0008,
        RightUp = 0x0010,
        MiddleDown = 0x0020,
        MiddleUp = 0x0040,
        VirtualDesk = 0x4000,
        Wheel = 0x0800,
        XDown = 0x0080,
        XUp = 0x0100
    }


    [DllImport("user32.dll", SetLastError = true)]
    private static extern uint SendInput(uint nInputs, Input[] pInputs, int cbSize);
    [DllImport("user32.dll")]
    private static extern IntPtr GetMessageExtraInfo();
    bool pressKey = false;
```

```
        public moddingTutorial()
        {
            this.Tick += onTick;
            this.KeyUp += onKeyUp;
            this.KeyDown += onKeyDown;


            switchLocations.Add(new Vector3(-1578.27f, 5155.2f, 19.79865f)); //Submari
            switchLocations.Add(new Vector3(82.81281f, 6432.408f, 31.31271f)); //
            switchLocations.Add(new Vector3(-237.6f, 5497.5f, 189.6f)); //

            index = 0;
        }

        private void checkDanger()
        {
            if (Game.Player.Character.IsSwimming || Game.Player.Character.IsInCombat |
            {
                //teleport to one of the locations
                Function.Call(Hash.SET_ENTITY_COORDS, Game.Player.Character, switchLoca

                //Move the index to the next location
                index++;
                if (index >= switchLocations.Count) index = 0;
#if (debug)
                UI.Notify("Teleport");
#endif
            }
        }

        private void onTick(object sender, EventArgs e) //this function gets executed
        {
            checkDanger();

            if (botOn)
            {
                //SET PLAYER INVINCIBLE
                Function.Call(Hash.SET_PLAYER_INVINCIBLE, Game.Player, true);

                //SET 1st PERSON VIEW
                Function.Call(Hash.SET_FOLLOW_PED_CAM_VIEW_MODE, 4);

                //TASK SEQUENCE
                TaskSequence mySeq = new TaskSequence();
```

```
                //WALK
                Random rndWalk = new Random();
                int Walk = (int)rndWalk.Next(-10, 10);
                //Game.Player.Character.Task.RunTo(Game.Player.Character.GetOffsetInWorldCoords(
                mySeq.AddTask.GoTo(Game.Player.Character.GetOffsetInWorldCoords(new Vector3(Walk,

                //RUN
                Random rndRun = new Random();
                int Run = (int)rndRun.Next(-10, 10);
                //Game.Player.Character.Task.RunTo(Game.Player.Character.GetOffsetInWorldCoords(
                mySeq.AddTask.RunTo(Game.Player.Character.GetOffsetInWorldCoords(new Vector3(0, R

                //WANDER AROUND
                //Game.Player.Character.Task.WanderAround();
                mySeq.AddTask.WanderAround();

                mySeq.Close();
                Game.Player.Character.Task.PerformSequence(mySeq);

                //LET IT WANDER FOR RANDOM TIME
                Random rndWandering = new Random();
                int Wandering = (int)rndWandering.Next(13, 25);
                Wait(Wandering*1000);
                Game.Player.Character.Task.ClearAllImmediately();
                checkDanger();
                Wait(500);

                if (Game.Player.Character.IsStopped)
                {
                    //PHOTO TAKING STUFF (VIRTUAL KEYS)

#if (debug)
                    UI.ShowSubtitle("take out phone", 1000);
#endif
                    keyOut(0xC8, 1000); //Arrow UP
                    Wait(1000);
#if (debug)
                    UI.ShowSubtitle("move down", 1000);
#endif
                    keyOut(0xD0, 500); //Arrow DOWN
                    Wait(1000);
#if (debug)
                    UI.ShowSubtitle("take left", 1000);
#endif
                    keyOut(0xCB, 500); //Arrow LEFT
```

```
                    Wait(1000);
#if (debug)
                    UI.ShowSubtitle("open app", 1000);
#endif
                    keyOut(0x1C, 500); //ENTER
                    Wait(3000);

                    //RANDOM ZOOM
                    //ZOOM IN
                    Random rndZoomIn = new Random();
                    int ZoomIn = (int)rndZoomIn.Next(4, 14);
#if (debug)
                    UI.ShowSubtitle("Zooooooming In " + ZoomIn + " times", 2000);
#endif
                    for (int i = 0; i < ZoomIn; i++)
                    {
                        mouseWheelOut(130);
                        Wait(20);
                    }
                    Wait(1000);
                    //ZOOM OUT
                    Random rndZoomOut = new Random();
                    int ZoomOut = (int)rndZoomOut.Next(2, 8);
#if (debug)
                    UI.ShowSubtitle("Zooooooming Out " + ZoomOut + " times", 2000);
#endif
                    for (int i = 0; i < ZoomOut; i++)
                    {
                        mouseWheelOut(unchecked((uint)-130));
                        Wait(20);
                    }
                    Wait(3000);

                    //TO DO: SMOOTH RANDOM MOUSE XY
                    /*Random rndCamMove = new Random();
                    int camMove = rndCamMove.Next(5, 10);
                    UI.ShowSubtitle("find the right framing", 1000 + (20 * camMove));
                    for (int i = 0; i < camMove; i++)
                    {
                        Random rndX = new Random();
                        Random rndY = new Random();
                        int newX = rndX.Next(-10, 10);
                        int newY = rndY.Next(-10, 10);
                        //mouseOut(newX*2, newY*2); //mouse xy
                        mouseOut(newX * 10, 0); //mouse xy
```

```
                    Wait(500);
                }
                Wait(2000);*/

                //CHOOSE A RANDOM FILTER
                Random rnd = new Random();
                int filter = rnd.Next(14);
#if (debug)
                UI.ShowSubtitle("choose a random filter", 1000 + (500 * filter));
#endif
                for (int i = 0; i < filter; i++)
                {
                    keyOut(0xD0, 500); //Arrow DOWN
                    Wait(500);
                }
                Wait(1000);

                //TAKE PHOTO
#if (debug)
                UI.ShowSubtitle("take photo", 1000);
#endif
                keyOut(0x1C, 500); //ENTER
                Wait(5000);

                //DELETE PHOTO
#if (debug)
                UI.ShowSubtitle("delete photo", 1000);
#endif
                keyOut(0xD3, 500); //DELETE
                Wait(2000);

                //CLOSE THE CAMERA APP
#if (debug)
                UI.ShowSubtitle("close app", 1000);
#endif
                keyOut(0x0E, 500); //BACK SPACE
                Wait(2000);

                //PUT AWAY THE PHONE
#if (debug)
                UI.ShowSubtitle("put away phone", 1000);
#endif
                keyOut(0x0E, 500); //BACK SPACE

                Wait(2000);
```

```
            }
        }
    }


    //VIRTUAL MOUSE X AND Y MOVEMENT
    public static void mouseOut(int coordX, int coordY)
    {
        //mouse move xy
        Input[] inputs = new Input[]
        {
            new Input
            {
                type = (int) InputType.Mouse,
                u = new InputUnion
                {
                    mi = new MouseInput
                    {
                        dx = coordX,
                        dy = coordY,
                        dwFlags = (uint)(MouseEventF.Move),
                        dwExtraInfo = GetMessageExtraInfo()
                    }
                }
            }
        };
        SendInput((uint)inputs.Length, inputs, Marshal.SizeOf(typeof(Input)));
    }

    //VIRTUAL MOUSE WHEEL SCROLLING
    public static void mouseWheelOut(uint duration)
    {
        //mouse wheel
        Input[] inputs = new Input[]
        {
            new Input
            {
                type = (int) InputType.Mouse,
                u = new InputUnion
                {
                    mi = new MouseInput
                    {
                        dx = 100,
                        dy = 100,
                        //time = 0,
```

```
                        mouseData = duration,
                        dwFlags = (uint)(MouseEventF.Wheel),
                        dwExtraInfo = GetMessageExtraInfo()
                    }
                }
            }
        };
        SendInput((uint)inputs.Length, inputs, Marshal.SizeOf(typeof(Input)));
    }

    //VIRTUAL KEY PRESS AND RELEASE
    public static void keyOut(ushort hexKey, int delay)
    {
        //key pressed
        Input[] inputs = new Input[]
        {
            new Input
            {
                type = (int)InputType.Keyboard,
                u = new InputUnion
                {
                    ki = new KeyboardInput
                    {
                        wVk = 0,
                        wScan = hexKey, //0xC8,
                        dwFlags = (uint)(KeyEventF.KeyDown | KeyEventF.Scancode),
                        dwExtraInfo = GetMessageExtraInfo()
                    }
                }
            }
        };


        //key released
        Input[] inputs2 = new Input[]
        {
            new Input
            {
                type = (int)InputType.Keyboard,
                u = new InputUnion
                {
                    ki = new KeyboardInput
                    {
                        wVk = 0,
                        wScan = hexKey, //0xC8,
```

```
                            dwFlags = (uint)(KeyEventF.KeyUp | KeyEventF.Scancode),
                            dwExtraInfo = GetMessageExtraInfo()
                        }
                    }
                }
            };

            //send key press / delay / release
            SendInput((uint)inputs.Length, inputs, Marshal.SizeOf(typeof(Input)));
            Wait(delay);
            SendInput((uint)inputs2.Length, inputs2, Marshal.SizeOf(typeof(Input)));

        }

        private void onKeyDown(object sender, KeyEventArgs e)
        {

        }
        private void onKeyUp(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.H)
            {
                //SWITCH THE BOT ON/OFF
                botOn = !botOn;
                if(botOn) UI.ShowSubtitle("GTA V Photographer Bot is ON", 3000);
                 else UI.ShowSubtitle("GTA V Photographer Bot is OFF", 3000);

            }
            if (e.KeyCode == Keys.J)
            {
                //TELEPORT MANUALLY
                Function.Call(Hash.SET_ENTITY_COORDS, Game.Player.Character, switchLocati
                index++;
                if (index >= switchLocations.Count) index = 0;
             }
        }
    }
}
```

FiveM

FiveM is a modification for Grand Theft Auto V that enables people to play multiplayer on customized dedicated servers.

- Go to fivem.net and download the client application.
- Run FiveM.exe. If you run the installer in an empty folder, FiveM will install there. Otherwise, it will install in `%localappdata%\FiveM`.
- Start FiveM from your Windows start menu.
- Click on Play and choose a Server to connect to

//start a server

//scripting for fivem https://docs.fivem.net/docs/scripting-manual/runtimes/csharp/

test video embed

Content Replication Assignment