02

플레이어 위치 매핑

많은 실세계 게임의 중심은 위치 기반 지도다. 실세계의 플레이어 위치와 가상 세계가 합쳐지면서 게임의 가상 요소들을 실세계에 반영한다. 플레이어는 주변 세상을 새로운 시각과 모험을 즐기는 마음가짐으로 탐험할 수 있다.

이 장에서는 유니티에 지도를 통합하는 여정을 위한 첫걸음을 뗀다. 하지만 유니티로 작업하기 전에 GIS와 GPS의 기본 개념을 알아볼 것이다. 간단한 용어들을 정의하고 기본을 이해함으로써 추후에 등장할 좀 더 복잡한 주제들을 다룰 기반을 마련할 수 있다. 그런 후 유니티를 사용해 Foody GO 프로젝트에 위치 기반 지도, 기본 캐릭터, 자유롭게 움직이는 카메라를 추가할 예정이다. 이 장에서 다루는 개념들은 고급 수준이지만, 간결한 구성을 위해 아직까지는 코드에 직접 손대지 않을 것이다. 혹 GIS를 이미 이해하고 있다면, 자유롭게 코드를 들여다봐도 좋다.

이 장에서는 다음과 같은 주제들을 다룬다.

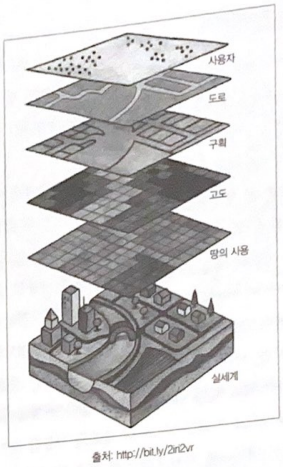
- GIS 용어들과 핵심 개념들
- GPS 용어들과 핵심 개념들
- 구글 맵스Google Maps
- 에셋 가져오기
- 서비스 구성
- CUDLR을 통한 디버깅

GIS 핵심 개념

GIS는 geographic information system의 약자로서 지도의 역할을 하는 지리적 데이터가 수집, 저장, 처리, 분석, 조종되는 시스템이다. 이 정의는 너무 광범위하지만, 근래에 GIS의 정의는 소프트웨어 애플리케이션, 하드웨어, 툴, 과학, 서비스 모두를 아우르는 개념으로 성장해왔다. 예를 들어 구글 맵스는 근대 GIS 활용의 가장 좋은 예제라고 할 수 있다. 이 책에서는 그중 GIS를 지리적 데이터와 매핑을 변환하는 과학적인 수단으로 사용할 것이다.

매핑

지도 제작자들은 1,000년이 넘는 시간 동안 지도를 만들어왔다. 하지만 근래에 들어서야 컴퓨터의 발전과 함께 GIS를 활용해 효율적으로 동적인 지도를 만들 수 있게 됐다. 손으로 그린 전통적인 지도와는 다르게 동적으로 그려진 지도의 경우 층layer으로 구성된 영역 데이터가 길, 주요 장소, 공원, 경계선, 지형, 물 등을 구성하고 있다. 구글 맵스의 사용자들은 자신이 볼 수 있는 층을 개별적으로 선택할 수는 없지만, 그 안에 분명히 존재하고 있다. 다음 그림은 도로 지도를 구성하는 일반적인 층들을 보여준다.



출처: http://bit.ly/2iri2vr

구글 맵스, 빙Bing과 같은 GIS 제공자들은 흔히 지도를 여러 개의 줌 단계로 그려낸 후 그려진 지도를 정적인 이미지 타일로 잘라낸다. GIS 서버는 이 타일들을 사용자에게 보여준다. 이 방식은 성능 면에서 효과적이지만, 도형이나 라인, 주요 지점 추가를 제외한 커스터마이징customizing과 스타일 변경 등의 기능은 제한적이다. 이런 매핑 방식을 보통 정적 매핑static mapping이라 부른다. 이 책에서는 동적 지도를 제공하는 Google Maps API를 사용할 것이다.

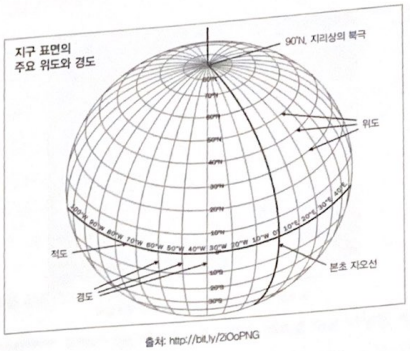
동적 지도는 개발자, 혹은 제공자로 하여금 데이터의 스타일과 기호화를 조절할 수 있는 옵션을 제공한다. 예를 들어 특정 지도에서 공원이 초록색이 아닌 파란색으로 보이길 원한다면 변경이 가능하고, 이런 요소들이 동적 지도가 제공하는 유연성이다. 이 장 후반에서 Google Maps API를 추가하면서 사용자 스타일과 기호화 옵션들을 다룰 예정이다.

GIS 매핑의 기본을 이해했으니 이제 용어와 개념을 파고들어보자. 다음은 지도를 제작하

거나 설명할 때 쓰이는 용어들이다.

- 지도 스케일map scale: 지리적 지도는 자신의 동네부터 전 세계까지 모두 보여줄 수 있다. 일반적으로 지도가 담고 있는 범위를 나타내기 위해 문자나 그림을 사용한다.
- 줌 레벨zoom level: 줌 레벨은 지도 스케일과 반대의 관계를 이루는 개념이다. 줌 레벨의 시작인 1은 전 세계를 보는 글로벌 뷰global view, 줌 레벨 17은 동네를 보여주는 지도를 말한다. 우리 게임에서는 주위 랜드마크를 쉽게 인지할 수 있는 줌 레벨 17이나 18 정도의 작은 스케일을 가진 지도를 사용할 것이다.
- 좌표 시스템coordinate system: 우리 역사를 보면, 주요 지점들을 표시하기 위해 다양한 좌표 시스템이 사용돼왔다. 많은 사람들이 위도와 경도를 사용하는 좌표 시스템은 알고 있지만, 다른 타입의 좌표 시스템이 있다는 것을 모르는 경우가 많다. 우리가 표준이라 알고 있는 위도와 경도도 많은 변형들이 있다. 이 책에서는 구글 맵스를 사용하므로 우리는 WGS 84 좌표를 사용할 것이다. 만일 본인이 다른 GIS를 사용해서 데이터를 가져오려고 할 경우 차이가 생기기 때문에 변환이 필요할 수 있다.

다음 그림은 WGS 84의 주요 라인들을 보여준다.



출처: http://bit.ly/2iOoPNG

본초 자오선 왼쪽에 있는 경도는 음수로 측정한다. 경도의 남쪽에 있는 위도도 음수로 측정한다.

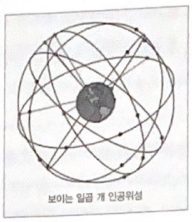
- 지도 투영map projection: 지도를 그릴 때 가장 근본적인 문제 중 하나는 3D로 구성돼 있는 세계를 2D로 표현하는 것이다. 초기 지도 제작자들은 이 문제를 해결하기 위해 구체에서 나온 빛을 원통형의 종이에 투영한 후 윤곽선을 따라 그리는 방법을 사용했다. 그런 후 종이를 펴면 2D로 된 지구의 모습이 생기는 것이다. 현재도 이와 유사한 방법을 사용해서 지도를 그리고 있다. 양극으로 갈수록 지도가 일그러지는 문제가 있지만, 일반적인 지도로서는 널리 사용되고 있다. 필요한 용도에 따라 지도를 투영하는 다양한 방법들도 존재한다. 양극의 일그러짐을 완화한 겔 피터스Gall-Peters와 같은 좀 더 발전된 투영 방식도 개발됐다. 하지만 우리는 구글 맵스의 기준을 따라 구글 웹 메르카토르Google Web Mercator, 줄여서 웹 메르카토르를 사용할 것이다. 다음 그림은 웹 메르카토르 투영 방식(왼쪽)과 겔 피터스 투영 방식(오른쪽)의 세계 지도를 비교한 것이다.

웹 메르카토르(왼쪽)와 겔 피터스(오른쪽) 투영 방식

GPS 핵심 개념

GPS는 Global Positioning System의 약자로, 매 시간마다 지구를 두 바퀴씩 공전하는 24개에서 32개의 인공위성 네트워크를 말한다. 공전하는 인공위성들은 지리적 신호를 시간 정보와 함께 송출하며, 인공위성을 볼 수 있는 위치에 있는 지구상의 GPS 기기들이 이를 감지한다. 기기가 인공위성으로부터 받아들이는 신호의 개수가 많을수록 위치의 결과는 더 정확해진다. GPS 삼각 측량과 정확도에 대해서는 4장, '포획물 생성하기'에서 더 자세히 다룬다.

다음 그림은 인공위성 네트워크에 속한 위성들 중 GPS가 볼 수 있는 위성으로부터 신호를 받는 모습이다.



보이는 인공위성을 추적하는 GPS 기기

다음은 GPS를 논의하거나 사용할 때 나올 수 있는 용어들이다.

- 데이텀datum: GPS가 인공위성으로부터 받은 신호를 사용 가능한 좌표로 변환할 때 사용하는 좌표 변환 시스템을 칭하는 용어다. 모든 GPS 기기는 WGS 84를 표준으로 삼고 있으며, 우리가 사용할 지도도 WGS 84를 사용한다. 전문 GPS 기기는 사용자의 필요에 맞게 여러 가지 다른 데이터 유형도 지원한다.
- 위도/경도latitude/longitude: 기본적으로 GPS 기기는 WGS 84 데이텀 형식으로 위도와 경도 좌표를 출력한다. 우리의 경우, 기기의 위치를 표시하기 위해 별도의 수학적인 변환이 필요 없으므로 편리하다.
- 고도altitude: 기기의 위치가 해수면에서 얼마나 떨어져 있는지를 의미한다. 우리가 게임을 실행할 대부분의 모바일 GPS 기기는 고도를 지원하지 않는다. 따라서 게임에서도 고도를 지원하지 않겠지만, 미래에는 지원됐으면 하는 바람이 있다.
- 정확도accuracy: 기기가 위치를 판단할 때 생기는 오차의 범위를 나타낸다. 신호를 받아들이는 인공위성의 숫자가 많을수록 위치의 계산이 더 정확해진다. GPS 기기는 물론, 일반인들에게 제공되는 GPS 네트워크는 정확도의 한계를 가지고 있다. 현대의 스마트폰은 보통 5~8미터의 정확도를 보이지만, 구형 스마트폰의 경우 최대 75미터까지 정확도 차이가 날 수 있다. 이 책 후반에서 플레이어와 가상 오브젝트의 상호작용을 다룰 때 GPS 정확도에 대해 좀 더 알아본다.

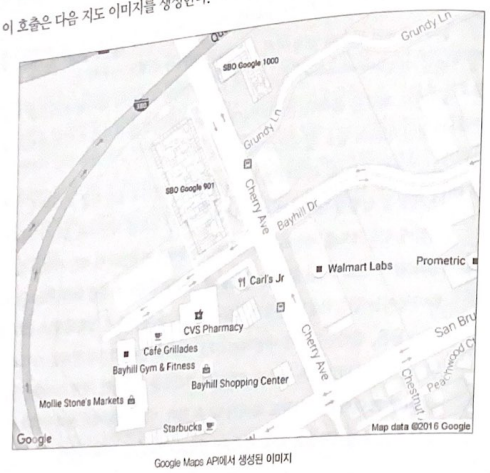
구글 맵스

이전에 언급한 바와 같이 우리가 사용할 지도 서비스는 구글 맵스Google Maps다. 이 게임은 구글 맵스의 고정 지도static map를 사용하므로, API 개발자 키developer key나 사용처에 대해 걱정하지 않아도 된다.

고정 지도 API를 사용하려면 일반적인 REST 서비스를 호출하는 방식과 유사하게 URL 안에 여러 개의 쿼리 문자열 파라미터query string parameter를 담은 GET 요청을 호출한다. 그러면 Google Maps API는 요청에 따라 하나의 이미지를 보내온다. 다음은 Google Maps API 고정 지도 REST 서비스를 호출하는 예제다.

https://maps.googleapis.com/maps/api/staticmap?center=37.62761,-122.42588&zoom=17&format=png&sensor=false&size=640x480&maptype=roadmap

이 호출은 다음 지도 이미지를 생성한다.



Google Maps API에서 생성된 이미지

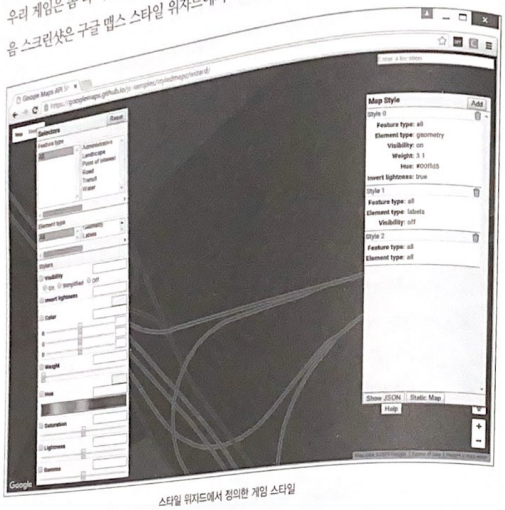
그냥 넘어가지 말고 반드시 위 링크를 클릭하거나, URL을 브라우저에 복사한 후 붙여 넣어 테스트해보자. 이제 URL을 부분별로 분리해서 지도를 요청할 때 어떤 요소들이 필요한지 알아보자.

- https://maps.googleapis.com/maps/api/staticmap: 구글 맵스 서비스의 기본 URL이다. 파라미터 없이 이 URL을 호출하면 에러가 발생한다. 이제 각 파라미터와 쿼리 구문query syntax을 자세히 들여다보자.
- ?: 물음표는 쿼리 파라미터의 시작을 알린다.
- * `center=37.62761,-122.42588`: 요청한 지도의 중심점을 위도와 경도 좌표로 나타낸다.
- &: 앰퍼샌드ampersand 심볼은 새로운 쿼리 파라미터의 시작을 말한다.
- `zoom=17`: 지도가 그려지는 줌 레벨 혹은 스케일을 말한다. GIS 핵심 개념을 다룰 때 언급했던 것처럼 줌 레벨이 높을수록 지도 스케일은 작아진다.
- `format=png`: 그려지는 이미지의 타입을 의미한다. 우리가 원하는 포맷은 PNG다.
- `sensor=false`: 위치를 GPS로 구한 것이 아니라는 의미다. 추후에 모바일 기기의 GPS를 사용하게 되면 `true`로 설정할 것이다.
- `size=640x480`: 이미지의 사이즈를 의미한다.
- `maptype=roadmap`: 지도의 타입이다. 요청할 수 있는 지도는 다음과 같이 모두 네 가지가 있다.
  - `roadmap`: 길, 대중교통, 영역 정보, 물, 주요 지점들을 보여주는 지도
  - `satellite`: 인공위성 사진을 보여주는 지도
  - `terrain`: `roadmap`과 고도를 같이 보여주는 지도
  - `hybrid`: `roadmap`과 `satellite` 지도를 같이 보여주는 지도

다행스럽게도 이 장에 같이 제공되는 스크립트가 위 URL을 생성해주기 때문에 사용자가 직접 생성할 필요는 없다. 하지만 지도 요청이 이뤄지는 방식을 알면 게임을 수정하거나 문제가 생겼을 때 큰 도움이 된다.

매핑을 다룰 때 GIS 지도는 언제나 레이어layer로 구성됐음을 기억할 것이다. 구글 맵스의 좋은 점은 요청 과정에서 맵 레이어의 스타일을 동적으로 변경할 수 있다는 것이다. 이 기능을 사용하면 게임에 어울리는 비주얼과 느낌을 만들어낼 수 있다. 잠시 시간을 할애해서 구글 맵스의 스타일 위자드style wizard(https://googlemaps.github.io/js-samples/styledmaps/wizard/)를 가지고 놀아보자.

우리 게임은 좀 더 어두운 분위기를 자아내기 위해 몇 가지 간단한 스타일을 적용했다. 다음 스크린샷은 구글 맵스 스타일 위자드에서 적용한 스타일을 보여준다.



스타일 위자드에서 정의한 게임 스타일

당장은 위자드에서 스타일을 뽑아 지도에 적용시키는 방식을 다루지 않겠다. 게임 수정 작업은 9장, '게임 마무리하기'에서 진행할 것이다. 궁금한 독자들은 스타일 위자드 왼쪽 패널에 있는 Static Map 버튼을 눌러서 스타일 파라미터들이 어떻게 구성되는지 볼 수 있다.

지도 추가하기

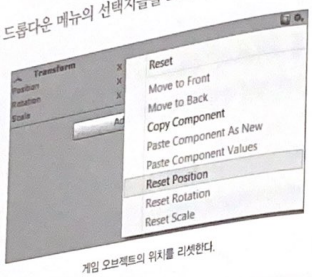
이제 기본적인 GIS와 GPS의 개념을 살펴보고 매핑에 대한 소개를 마쳤으니 유니티로 돌아가서 게임에 지도를 추가해보자. 지도를 구성하면서 이전에 언급했던 GIS 용어들을 다시 보게 될 것이다. 이전 장의 마지막 부분에서 다시 시작해보자.

맵 타일 생성하기

다음 설명을 따라 게임에 지도를 추가해보자.

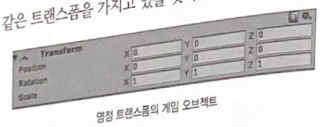
1. 유니티를 열고 이전 장에서 만든 FoodyGO 프로젝트를 불러오자. 만일 이전 장을 건너뛰고 바로 건너왔다면 다운로드한 소스 코드에서 프로젝트를 불러와도 된다. Chapter_2_Start 폴더를 열어 유니티로 프로젝트를 불러온다.
2. 유니티가 열리면 Splash 스크린이 보일 것이다. 만일 보이지 않는다고 해도 새로운 신scene을 만들 예정이므로 문제없다. 메뉴로 가서 File > New Scene을 선택한다.
3. Main Camera와 Directional Light만 있는 새로운 신이 생성될 것이다. 잊어버리기 전에 새로 만든 신을 저장하자. 메뉴에서 File > Save Scene as…를 선택한 후 Save Scene 창에서 파일 이름으로 Map을 입력하고 Save를 클릭한다.
4. Hierarchy 창에서 Map 신을 선택한다. 메뉴에서 GameObject > Create Empty를 선택해 신에 새로운 빈 GameObject를 생성한다. 새로운 게임 오브젝트를 선택하고 Inspector 창에 나오는 속성들을 둘러본다.
5. Inspector 창의 이름 필드에서 GameObject라는 이름을 `Map_Tiles`로 변경한다. Transform 컴포넌트에 있는 기어 아이콘을 클릭하면 드롭다운 메뉴에 나오는 Reset Position을 클릭해서 오브젝트의 트랜스폼 위치transform position를 리셋한다.

다음 스크린샷은 드롭다운 메뉴의 선택지들을 보여준다.



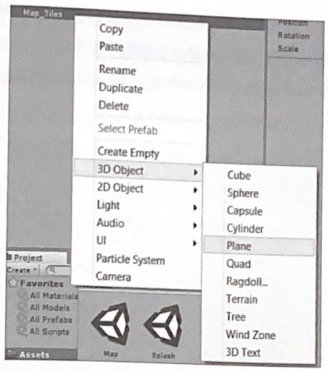
게임 오브젝트의 위치를 리셋한다.

1. GIS 수학적 변환을 단순화하기 위해 대부분 게임 오브젝트들의 위치를 영점zero이나 영점에 가깝게 리셋시킬 것이다. 현재 Map_Tiles 게임 오브젝트는 다음 스크린샷과 같은 트랜스폼을 가지고 있을 것이다.



영점 트랜스폼의 게임 오브젝트

2. Map_Tiles 게임 오브젝트가 선택된 상태에서 오른 클릭(맥에서는 command + 클릭)을 해서 연관 메뉴를 열고 3D Object > Plane을 선택한다. 다음 스크린샷은 연관 메뉴에서 플레인plane을 선택하는 방법을 보여준다.

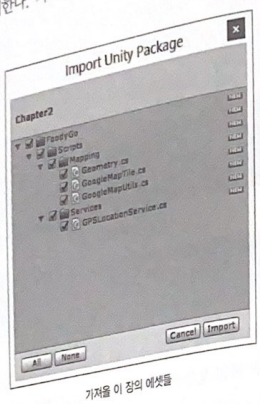


게임 오브젝트의 연관 메뉴

3. 플레인 게임 오브젝트를 선택하고 Inspector 창에서 Map_Tile로 이름을 변경한다. 오브젝트의 트랜스폼이 영점에 있는지 확인한다.
4. Hierarchy 창의 Map_Tile 플레인을 더블 클릭해서 Scene 창의 중심에 오브젝트가 위치하게 한다. 만일 오브젝트가 보이지 않으면 Scene 창의 2D 버튼이 꺼져 있는지 확인한다.
5. Inspector 창에서 Transform 컴포넌트에 있는 X와 Z의 스케일scale을 10으로 변경한다. 스케일을 변경할 때 플레인의 크기가 변하는 것을 눈여겨보자.
6. 이제 Map_Tile 오브젝트에 스크립트를 첨부해서 지도를 그릴 수 있게 해보자. 아직은 새로운 스크립트를 직접 생성하는 것을 피하고, 가져올import 에셋에 들어있는 스크립트를 첨부해보자. 물론 책 후반부에서는 새로운 스크립트를 직접 생성할 것이다. 우리가 필요한 스크립트는 다운로드한 코드 안에 있는 Chapter_2_Assets 폴더에서 찾을 수 있다. 메뉴에서 Assets > Import Package > Custom Package…를 선택해서 Import package 창을 연다.

7. 다운로드한 소스 코드의 Chapter_2_Assets 폴더로 가서 Chapter2.unity package를 선택하고 Open을 클릭해 가져온다.

8. 에셋을 가져오는 진행 상황을 보여주는 창이 나오고 나면 Import Unity Package 창이 뜬다. 창에 있는 모든 아이템들이 선택돼 있고 가져올 수 있는지 확인한 후 Import를 클릭한다. 가져올 스크립트가 들어있는 Import Unity Package 창이 열릴 것이다.



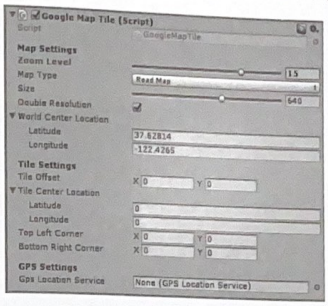
가져올 이 장의 에셋들

9. 에셋을 가져온 후 Project 창을 보면 Assets 안에 새로운 폴더가 생긴 것을 볼 수 있다. 이 폴더의 내용을 자유롭게 둘러보고, 프로젝트 에셋들이 어떻게 정리돼 있는지 숙지해두자. 우리가 가져온 그대로 폴더 구조가 구성돼 있는 것도 눈여겨보자.

10. Hierarchy 창에 있는 Map_Tile 오브젝트를 선택한다. Inspector 창 하단에 있는 Add Component 버튼을 클릭한다. 컴포넌트 리스트를 보여주는 연관 메뉴가 열릴 것이다. Mapping > Google Tile Map을 선택한다. Map_Tile 게임 오브젝트에 구글 매핑 스크립트가 첨부될 것이다.

유니티 에셋 스토어Unity Asset Store에서 무료로 제공하는 Google Maps for Unity에서 영감을 받아 Google Tile Map 코드를 만들었다. 게임에 사용할 수 있는 고급 버전을 만들기 위해 여러 부분을 변경했다.

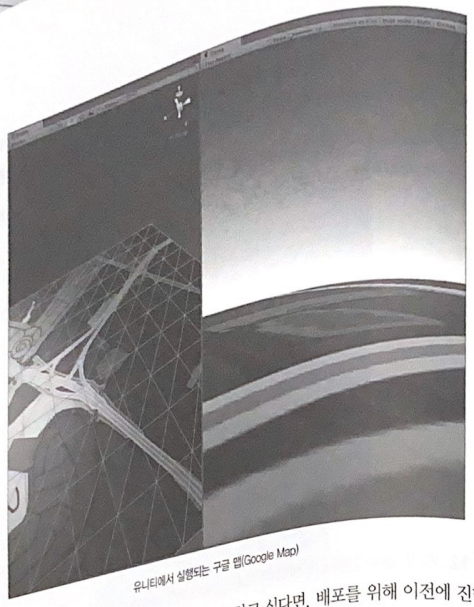
11. Inspector 창에서 Google Map Tile 스크립트를 다음과 같이 수정한다.



위와 똑같이 컴포넌트의 값을 수정한다.

12. 이 장 앞부분에서 설명했던 내용을 바탕으로 위 매핑 파라미터의 의미를 어느 정도 이해했길 바란다. 일단 지금은 Zoom Level을 15로 설정해서 지도가 어떻게 작동하는지 테스트해보자. 좌표는 샌프란시스코에 있는 구글 본사의 위치다. 물론 추후에는 기기의 GPS를 연결해서 유저의 좌표를 사용할 예정이다.

13. Play 버튼을 누른다. 몇 초가 지나면 다음과 비슷한 화면이 나올 것이다.



유니티에서 실행되는 구글 맵(Google Map)

14. 본인의 모바일 기기에서 게임을 테스트하고 싶다면, 배포를 위해 이전에 진행했던 단계들을 다시 하면 된다. 게임을 배포하는 방법이 정확히 기억나지 않는다면 1장, '시작하기'를 참조하자.

본인 기기에 이미 게임을 배포해봤다고 가정한 후, 배포 단계를 다음과 같이 요약했다.

1. 유니티 개발 컴퓨터와 본인의 기기가 USB 케이블로 연결돼 있는지 확인한다.
2. File > Save Scene을 선택해 신을 저장한다.
3. File > Save Project를 선택해 프로젝트를 저장한다. 게임을 빌드하기 전에 게임을 저장하는 습관을 들이자. 유니티 빌드 과정이 에디터를 크래시crash시키는 일이 간혹 일어난다.

4. 메뉴에서 File > Build Settings…를 선택해서 Build Settings 창을 연다.
5. 지금은 불필요하니 Splash 신의 체크를 해제한다. Add Open Scenes 버튼을 클릭해서 빌드에 Map 신을 추가한다.
6. 본인에게 맞는 배포 플랫폼(안드로이드 혹은 iOS)을 선택한다.
7. Build and Run 버튼을 클릭해서 빌드와 배포 과정을 시작한다.
8. 알림이 나오면 이전에 선택했던 것과 같은 위치에 배포 파일을 저장하고 덮어쓴다.
9. 게임이 빌드될 때까지 기다린 후 기기에 배포한다.
10. 본인의 기기에 게임이 불려왔으면, 지도를 확인하고 기기를 돌려보자. 지금은 게임에서 되는 것이 거의 없지만, 지도가 정상적으로 작동하고 있는 것은 알 수 있다.

처음 눈에 띄는 부분 중 하나는 지도 이미지가 더 밝게 보인다는 것이다. 라이팅 처리와 플레인plane의 기본 머티리얼 때문인데, 다행히도 이 스타일은 우리가 원하는 모습이기 때문에 좀 더 밝아진 상태 그대로 두기로 한다.

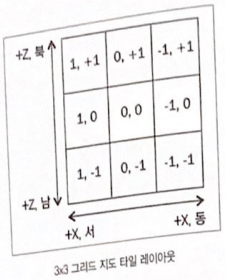
두 번째로 눈에 띄는 부분은 지도가 스타일 위자드의 서버에서 그려졌던 이미지보다 픽셀이 두드러져 보인다는 점이다. 이 현상은 플레인 위에 그려진 지도 이미지가 늘려져 있기 때문이다. 상식적인 해결책은 이미지의 사이즈와 해상도를 높이는 것이지만, 불행하게도 구글 맵스에 요청할 수 있는 최대 이미지 사이즈는 약 1200×1200픽셀이며, 이미 우리는 두 배의 해상도로 사용하고 있다. 따라서 더 깔끔하고 깨끗하게 보이는 지도를 위해 다른 방법을 찾아야 한다. 다음 절에서는 픽셀화 문제를 해결해본다.

타일 깔기

지도의 세부적인 묘사들과 가는 선들 때문에 우리 지도는 최대한 높은 해상도를 사용해서 그려지길 원한다. 하지만 높은 해상도의 이미지는 많은 자원을 사용하고 에러가 많이 나곤 한다. 다행히도 다른 사람들이 여러 개의 이미지, 혹은 이미지 타일들을 조합하는 방법

으로 해상도 문제를 해결한 여러 예제들이 있다.

우리도 같은 방법을 사용해서 우리 지도를 하나의 타일이 아니라 다음과 같은 3×3 타일 그리드grid로 확장해보자.



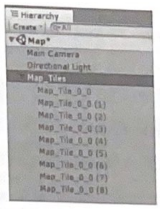
3x3 그리드 지도 타일 레이아웃

도표를 보면 x축과 타일 오프셋offset이 반대 방향의 관계임을 알 수 있다. 다시 말해, X 방향의 타일 오프셋은 3D 공간에서 x축으로 반대의 방향이라는 것이다. z축과 Y 타일 오프셋은 정방향의 관계다. Y 타일 오프셋이 1이라면, z축도 양수가 된다. 플레이어는 땅과 가까이 있기 때문에 3×3 그리드만 있어도 충분하다. 만일 좀 더 높은 위치의 카메라를 놓고 싶거나, 지평선을 좀 더 멀리 보여주고 싶다면 타일 레이아웃을 5×5, 7×7, 9×9 등 원하는 만큼 확장시킨다.

이제 우리 지도를 하나의 타일에서 3×3 그리드 타일로 확장시켜보자. 다음 설명을 따라 유니티에서 지도 타일 레이아웃을 구성해보자.

1. Hierarchy 창에서 Map_Tile 게임 오브젝트를 선택한다. Inspector 창에서 오브젝트 속성들을 다음과 같은 값으로 설정한다.
   - Transform, Scale, X: 3
   - Transform, Scale, Z: 3
   - Google Map Tile, Zoom Level: 17

2. FoodyGo 폴더 안에서 오른 클릭(맥에서는 command + 클릭)을 하고 연관 메뉴를 연다. Create > New Folder를 선택해서 새로운 폴더를 생성한다. 생성된 폴더의 이름을 Prefabs로 변경한다.
3. 이제 Map_Tile 오브젝트를 프리팹prefab으로 만들어보자. 프리팹은 게임 오브젝트의 복제, 혹은 템플릿template으로 이해하면 된다. 프리팹을 만들기 위해서는 Map_Tile 게임 오브젝트를 선택하고 새로 생성한 Prefabs 폴더로 끌어다 놓는다. 프리팹 폴더 안에 Map_Tile이라는 새로운 프리팹이 생성된 것을 볼 수 있다. 프리팹이 만들어지고 나면 Hierarchy 창에 있는 Map_Tile 게임 오브젝트가 파란색으로 변한 것을 볼 수 있다. 파란색은 해당 게임 오브젝트가 프리팹과 연결돼 있다는 뜻이다.
4. Hierarchy 창으로 돌아가서 Map_Tile 오브젝트를 선택하고 이름을 Map_Tile_0_0으로 변경한다. 이 오브젝트가 중앙(0, 0) 타일임을 의미하기 위해서다.
5. Hierarchy에서 Map_Tile_0_0 게임 오브젝트를 선택하고, Ctrl + D(맥에서는 command + D)를 눌러 지도 타일을 복제한다. 여덟 번 반복해서 다음과 같이 여덟 개의 추가 지도 타일을 만든다.

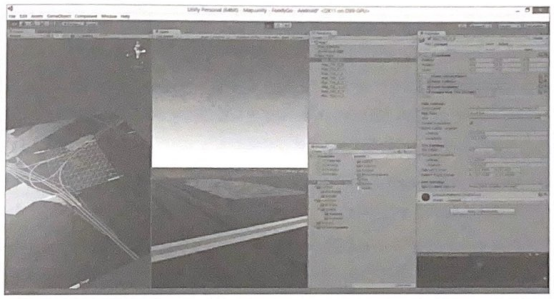


Map_Tiles 게임 오브젝트는 복제된 지도 타일들의 부모다.

6. 복제된 지도 타일들의 이름과 속성들을 Inspector 창에서 다음 표와 같이 변경한다.

게임 오브젝트	속성
Map_Tile_0_0 (1)	Name: Map_Tile_0_1 Transform.Position.X: 0 Transform.Position.Z: 30 GoogleMapTile.TileOffset.X: 0 GoogleMapTile.TileOffset.Y: 1
Map_Tile_0_0 (2)	Name: Map_Tile_0_−1 Transform.Position.X: 0 Transform.Position.Z: −30 GoogleMapTile.TileOffset.X: 0 GoogleMapTile.TileOffset.Y: −1
Map_Tile_0_0 (3)	Name: Map_Tile_1_0 Transform.Position.X: −30 Transform.Position.Z: 0 GoogleMapTile.TileOffset.X: 1 GoogleMapTile.TileOffset.Y: 0
Map_Tile_0_0 (4)	Name: Map_Tile_−1_0 Transform.Position.X: 30 Transform.Position.Z: 0 GoogleMapTile.TileOffset.X: −1 GoogleMapTile.TileOffset.Y: 0
Map_Tile_0_0 (5)	Name: Map_Tile_1_1 Transform.Position.X: −30 Transform.Position.Z: 30 GoogleMapTile.TileOffset.X: 1 GoogleMapTile.TileOffset.Y: 1
Map_Tile_0_0 (6)	Name: Map_Tile_−1_−1 Transform.Position.X: 30 Transform.Position.Z: −30 GoogleMapTile.TileOffset.X: −1 GoogleMapTile.TileOffset.Y: −1
Map_Tile_0_0 (7)	Name: Map_Tile_−1_1 Transform.Position.X: 30 Transform.Position.Z: 30 GoogleMapTile.TileOffset.X: −1 GoogleMapTile.TileOffset.Y: 1
Map_Tile_0_0 (8)	Name: Map_Tile_1_−1 Transform.Position.X: −30 Transform.Position.Z: −30 GoogleMapTile.TileOffset.X: 1 GoogleMapTile.TileOffset.Y: −1

7. Play 버튼을 눌러 게임을 실행시킨다. 게임이 실행되는 중에 Hierarchy 창에서 Map_Tile_0_0을 선택하고 F를 눌러 오브젝트를 Scene 창의 중앙에 위치시킨다. 지도의 픽셀화가 확연히 줄어든 것을 볼 수 있으며, 다음 스크린샷과 비슷한 모습일 것이다.



게임 실행 중 플레이 모드에서 본 타일화된 지도의 모습

코드 이해하기

이제 멋진 지도가 게임에 들어갔다. 물론 구성하는 과정이 조금 반복적이긴 하지만, 신중하게 진행했다면 그리 오래 걸리지 않았을 것이다. 타일을 배치하기 위해 많은 수학적 계산을 예상하고 있었다면, 다행히도 `GoogleMapTile` 스크립트에서 다 처리하고 있다. 이제는 유니티에서 잠시 벗어나 모노디벨롭MonoDevelop에서 `GoogleMapTile` 스크립트를 들여다보자.

유니티의 Hierarchy 창에서 Map_Tile_0_0을 선택한다. Inspector 창의 Google Map Tile 스크립트 컴포넌트에서 기어 아이콘을 클릭해 연관 메뉴를 연다. 메뉴에서 Edit Script를 선택한다. 몇 초 후에 모노디벨롭이 열릴 것이다.

> **TIP** 모노디벨롭은 유니티의 기본 스크립트 에디터다. 윈도우에서 개발하고 있다면 Visual Studio Community나 상급 버전도 훌륭한 선택이다. 또 다른 좋은 선택지로는 윈도우, 맥, 리눅스에서 모두 사용할 수 있으면서 가벼운 Visual Studio Code가 있다.

모노디벨롭에서 GoogleMapTile 스크립트가 열린 것을 볼 수 있다. 이 책의 시작 부분에서 언급한 것처럼, 독자는 C#에 대한 기본 지식이 있어야 하므로 스크립트의 내용이 필요 이상으로 복잡하게 느껴지지 않아야 한다. 만일 유니티 스크립팅이 처음이라도 큰 문제는 없으며, 추후에 스크립트 작성에 대해 좀 더 깊게 다룰 것이다. 일단 지금은 지도의 타일화를 작동하게 하는 코드의 특정 부분들에 집중해보자.

IEnumerator_RefreshMapTile() 함수가 나올 때까지 밑으로 내려보자. 다음은 우리가 눈여겨볼 메소드의 초반 부분이다.

```
IEnumerator _RefreshMapTile()
 {
    // 타일 중앙의 위도와 경도를 찾는다
    tileCenterLocation.Latitude = GoogleMapUtils.adjustLatByPixels(worldCenterLoc
ation.Latitude, (int)(size * 1 * TileOffset.y), zoomLevel);
    tileCenterLocation.Longitude = GoogleMapUtils.adjustLonByPixels(worldCenterLo
cation.Longitude, (int)(size * 1 * TileOffset.x), zoomLevel);
```

주석의 문장처럼 위 두 라인의 코드는 타일 중앙의 위도latitude와 경도longitude 지도 좌표를 찾는다. 위도는 타일 이미지 사이즈size를 TileOffsett.y를 곱해서 얻고, 경도는 TileOffset.x를 곱해서 얻는다. 곱셈의 결과와 zoomLevel은 GoogleMapUtils 헬퍼 함수로 전달돼, 조절된 타일의 위도 혹은 경도를 계산한다. 간단해 보이지 않는가? 물론 중요한 부분은 GoogleMapUtils 함수들이 처리하며, 이 함수들은 거리 변환을 위한 기본 GIS 수학 함수들이다. 궁금하다면 GoogleMapUtils 코드를 둘러보자. 하지만 지금은 _RefreshMapTile 메소드를 계속해서 다뤄본다.

다음 코드가 나올 때까지 계속 스크롤을 내려보자.

```
// 맵 타일을 요청하기 위한 쿼리 스트링 파라미터를 구성한다
queryString += "center=" + WWW.UnEscapeURL (string.Format ("{0},{1}",
tileCenterLocation.Latitude, tileCenterLocation.Longitude));
queryString += "&zoom=" + zoomLevel.ToString ();
queryString += "&size=" + WWW.UnEscapeURL (string.Format ("{0}x{0}", size));
queryString += "&scale=" + (doubleResolution ? "2" : "1");
queryString += "&maptype=" + mapType.ToString ().ToLower ();
queryString += "&format=" + "png";

// 맵 스타일을 추가한다
queryString += "&style=element:geometry|invert_lightness:true|weight:3.1|hue:0x00
ffd5";
queryString += "&style=element:labels|visibility:off";
```

주석의 설명처럼 코드의 이 부분은 Google Maps API에 지도 이미지를 요청할 때 필요한 쿼리 파라미터query parameter를 구성한다. 이 파라미터는 URL 형태로 전달되므로 특수 캐릭터들의 인코딩encoding을 확실히 해야 하며, WWW.UnEscapeURL 호출이 그 역할을 담당한다. 하단을 보면 몇 가지 스타일 추가도 볼 수 있다. 9장, '게임 마무리하기'에서는 구글 맵스 스타일 위자드Google Maps Style Wizard를 사용해 자신이 원하는 스타일을 쉽게 추가할 수 있는 방법을 알아볼 예정이다.

마지막으로 _RefreshMapTile 메소드의 하단으로 스크롤하자. 다음은 코드의 일부분이다.

```
// 마지막으로, 이미지를 요청한다
var req = new WWW(GOOGLE_MAPS_URL + "?" + queryString);
// 서비스가 응답할 때까지 기다린다
yield return req;
// 이전 텍스처를 제거한다
Destroy(GetComponent<Renderer>().material.mainTexture);
// 에러를 체크한다
if (req.error != null)
```

```
{
  print(string.Format("Error loading tile {0}x{1}: exception={2}", TileOffset.x,
TileOffset.y, req.error));
}
  else
  {
    // 에러가 없으면 이미지를 그린다
    // 이미지가 불려오면 타일 텍스처로 지정한다
    GetComponent<Renderer>().material.mainTexture = req.texture;
      print(string.Format("Tile {0}x{1} textured", TileOffset.x, TileOffset.y));
        }
```

첫 라인에서는 WWW 클래스를 이용해 이전에 만든 queryString과 GOOGLE_MAPS_URL을 연결해 요청을 구성한다. WWW 클래스는 유니티 헬퍼 클래스로서 URL로 모든 것을 호출할 수 있게 해준다. 책의 후반부에서는 이 클래스를 이용해 다른 서비스 요청도 만들 예정이다.

다음 줄 yield return req;는 유니티로 하여금 해당 요청을 반응이 올 때까지 지속하도록 한다. 이 메소드가 코루틴coroutine이기 때문에 가능한 방법이다. 코루틴은 IEnumerator를 반환하는 메소드로서 스레드 블로킹thread blocking을 예방하는 매끄러운 방법이다. C# 비동기 프로그래밍을 고전적인 방법으로 해본 경험이 있다면 코루틴에게 매우 감사한 마음이 들 것이다. 스크립트 작성에 본격적으로 뛰어들게 되면 코루틴에 대해 좀 더 자세히 설명한다.

이어서 오브젝트의 현재 텍스처에 Destroy를 호출한다. Destroy는 MonoBehaviour 클래스에 속해 있는 퍼블릭 메소드로서 오브젝트와 오브젝트에 첨부돼 있는 모든 컴포넌트들을 안전하게 제거할 수 있게 해준다. C# Windows를 이용해봤거나 웹 개발 경험이 있는 사람이라면 이 단계가 어색하게 느껴질 수도 있겠다. 게임이 실행되고 있는 동안 메모리 관리가 빠르게 감당하기 어려워질 수 있으므로 이 부분을 항상 염두에 둬야 할 것이다. 이 예제의 경우 해당 코드를 삭제하면 텍스처 메모리 누수texture memory leak 때문에 게임이 크래시crash될 확률이 매우 높다.

Destroy 호출 후에는 에러 체크를 통해 이미지 타일을 요청할 때 에러가 발생하지 않았는지 확인한다. 에러가 발생했다면 에러 메시지를 출력한다. 그렇지 않으면 현재 텍스처를 새로 다운로드한 이미지로 변경한다. 그런 후 print를 사용해서 Console 창에 디버그 메시지debug message를 출력한다. Print 메소드는 Debug.log를 호출하는 것과 똑같이 작동하지만 MonoBehaviour 클래스에서만 사용할 수 있다.

마지막으로 언제 _RefreshMapTile 메소드가 호출되는지 코드를 통해 이해해보자. 다음과 같이 Update 메소드가 보일 때까지 스크롤을 올려보자.

```
// Update is called once per frame
  void Update ()
  {
    // 새로운 위치가 얻어졌는지 확인한다
    if (gpsLocationService != null && gpsLocationService.IsServiceStarted &&
lastGPSUpdate < gpsLocationService.Timestamp)
    {
      lastGPSUpdate = gpsLocationService.Timestamp;
      worldCenterLocation.Latitude = gpsLocationService.Latitude;
      worldCenterLocation.Longitude = gpsLocationService.Longitude;
      print("GoogleMapTile refreshing map texture");
      RefreshMapTile();
    }
  }
```

Update는 특별한 유니티 메소드로서 MonoBehaviour 클래스에서 나온 것이다. 주석에서 설명한 것처럼 Update 메소드는 매 프레임frame마다 호출된다. 당연한 말이지만 요청의 답이 그렇게 빨리 돌아올 리 없으므로 매 프레임마다 지도를 새로 고칠 필요는 없다. 대신에 위치 서비스가 시작됐고, 사용하고 있는지 확인한다. 그런 후 타임스탬프timestamp 변수를 체크해서 위치 서비스가 움직임을 감지했는지 확인한다. 위 세 개의 테스트를 통과했으면 타임스탬프를 업데이트하고, 세계의 새로운 중앙점을 얻고, 메시지를 출력하고, 마지막으로 RefreshMapTile을 호출한다. RefreshMapTile은 StartCoroutine(_RefreshMapTile)

을 호출해서 타일의 새로 고침을 시작한다.

아직 GPS 서비스를 연결하지 않았기에 이 모든 작업이 어색하게 느껴질 수 있다. 하지만 곧 진행할 것이므로 걱정할 필요는 없다. 일단은 지도 타일이 얼마나 자주 새로 그려지는지 이해하는 것만으로도 큰 도움이 된다.

이 절에서는 단일 이미지가 아닌 이미지 타일을 그려서 게임의 지도 해상도를 높였다. 우리의 목적에 부합하기 위해 각 이미지 타일도 이미 꽤 큰 타일 사이즈이긴 하다. 하지만 카메라가 높은 곳에서 플레이어를 내려다보고 있으므로 큰 문제가 되지는 않는다. 위 작업을 보면 알겠지만 타일 지도를 생성하는 것은 사이즈에 상관없이 그다지 어려운 일이 아니다. 혹시 더 큰 사이즈의 지도를 만들고 싶다면, 많은 지도 타일을 다운로드하는 과정이 플레이어의 데이터 소모를 크게 늘릴 수 있다는 점을 염두에 두자.

서비스 구성하기

서비스service라는 용어는 애플리케이션이나 필요에 따라 매우 넓은 의미를 가진다. 우리의 경우 서비스는 게임 오브젝트가 사용하는 스스로 관리되고 실행되는 코드라고 정의하자. 서비스는 오브젝트, 혹은 오브젝트들의 형태로 실행되기 때문에 라이브러리library나, `GoogleMapUtils`와 같은 글로벌 스태틱 클래스global static class와는 다르다. 경우에 따라 싱글톤 패턴singleton pattern을 사용해서 서비스를 구현하고자 할 수도 있지만, 이 책에서는 최대한 단순한 코드를 사용하는 것이 목적이므로 게임 오브젝트를 사용해서 서비스를 생성하고 사용해본다.

이 장에서는 두 개의 서비스를 구성할 것이다. 플레이어의 위치를 구하기 위한 GPS 위치 서비스GPS Location Service와 디버깅을 위한 CUDLR이다. 먼저 CUDLR 서비스로 시작해보자. 이는 위치 서비스를 구성할 때 발생할지도 모를 문제를 디버깅할 때 도움이 된다.

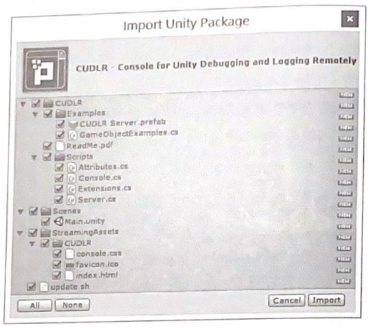
CUDLR 구성하기

CUDLR은 Console for Unity Debugging and Logging Remotely의 약자로, 유니티 에셋 스토어Unity Asset Store에서 무료로 제공되는 에셋이다. 게임이 실행될 때 기기의 행동을 주시하고, 원격으로 간단한 콘솔 명령을 실행하기 위해 CUDLR을 사용할 것이다. 다음 장에서는 또 다른 강력한 진단 툴인 유니티 리모트Unity Remote를 다뤄보겠지만, 유니티의 주장과는 달리 종종 위치 서비스를 접근하는 데 실패하는 경우가 있다. 게임 개발을 진행하다 보면, 게임을 원격으로 모니터링하고 조종할 수 있는 방법은 언제나 도움이 된다.

> TIP CUDLR을 사용하기 위해서는 본인의 기기와 개발 컴퓨터가 같은 로컬 와이파이(Wi-Fi) 네트워크상에 있어야 한다. 만일 모바일 기기가 로컬 와이파이 네트워크에 접속할 수 없다면 이 절을 건너뛴다.

다음 단계를 진행해서 CUDLR을 설치하자.

1. 메뉴에서 Window > Asset Store로 가서 에셋 스토어를 연다. 창이 열리면 Search 란에 `cudlr`을 입력하고 Enter를 누른다. 몇 초가 지나면 에셋 리스트가 나올 것이다.
2. CUDLR을 위한 이미지나 링크를 클릭하면 에셋 페이지가 나온다. 페이지가 열리면 Import 버튼을 클릭해서 에셋을 프로젝트로 가져온다.
3. 에셋이 매우 작은 편이라 다운로드는 빠르게 이뤄질 것이다. 다운로드가 끝나면 Import Unity Package 창이 열린다. 다음 스크린샷에서 보는 것처럼 모든 부분들이 선택돼 있는지 확인한다.



CUDLR 에셋 가져오기

4. 이 에셋들을 프로젝트에 처음 가져오기 때문에 모든 부분들을 다 설치한다. 이후 개발에서는 에셋들 중 Examples 폴더와 같이 필요 없는 부분을 언제든지 제거할 수 있다. 가져올 준비가 끝났으면 창에 있는 Import 버튼을 클릭하고 가져오기가 끝나길 기다린다.
5. 메뉴에서 GameObject ➤ Create Empty로 가서 부모 서비스 오브젝트 역할을 할 오브젝트를 신에 생성한다. Hierarchy 창에 새로운 빈 게임 오브젝트가 생길 것이다. 새로운 오브젝트의 이름을 Services로 변경한다.
6. Hierarchy 창에 있는 Services 오브젝트에 오른 클릭(맥에서는 Ctrl + 클릭)을 해서 연관 메뉴를 연다. 연관 메뉴에서 Create Empty를 선택한다. Services 오브젝트의 자식 오브젝트인 빈 게임 오브젝트가 GameObject라는 이름으로 생성될 것이다. 이 과정을 반복해서 빈 게임 오브젝트를 하나 더 생성한다.
7. 첫 번째 빈 GameObject를 선택하고 CUDLR로 이름을 변경한다. 그런 후 두 번째를 GPS로 이름을 변경한다. GPS 서비스는 나중에 추가할 예정이지만 이왕 작업 중이니 미리 만들어놓자.

8. Project 창에서 Assets/CUDLR/Scripts 폴더를 연다. Project 창에 있는 서버 스크립트server script를 끌어 Hierarchy 창에 있는 CUDLR 게임 오브젝트에 놓는다. 이렇게 하면 CUDLR 서버 컴포넌트가 게임 오브젝트에 추가된다. 이제 CUDLR이 준비됐다.

CUDLR로 디버깅하기

CUDLR이 훌륭한 툴인 이유는 게임의 일부분을 웹 서버로 만들어주기 때문이다. 이제 백도어backdoor를 심어놓은 것처럼 우리 게임을 살펴보고 통신할 수 있다. 또한 CUDLR은 네트워크에 있는 어느 컴퓨터에서도 접근할 수 있으므로, 게임을 제어하기 위해 직접 연결할 필요도 없고 유니티 프로그램을 별도로 실행할 필요도 없다. 물론 게임이 로컬 웹 서버 역할을 하는 것은 게임은 물론 플레이어의 기기에도 보안상 문제가 될 수 있다. 따라서 게임을 출시하기 전에는 CUDLR 서비스 게임 오브젝트를 삭제해서 비활성화할 것이다.

다음 설명을 따라 게임에서 실행되는 CUDLR 서비스에 연결해보자.

1. 모바일 기기를 열고 IP 주소를 찾아 적어놓거나 기억해둔다. 다음 단계를 따라 안드로이드나 iOS의 IP 주소를 찾는다.
   - 안드로이드: Settings ➤ About phone ➤ Status로 가서 IP Address 부분까지 내려간다.
   - iOS: WiFi를 선택해 현재 연결 가능한 무선 네트워크 리스트를 연다. 현재 연결돼 있는 무선 네트워크를 찾은 후 네트워크 이름 오른쪽에 있는 하얀색 화살표가 그려진 파란색 원을 탭한다. DHCP 탭 아래에 있는 첫 번째 필드가 IP Address일 것이다.
2. 이 장에서 배운 방법을 사용해 게임을 빌드하고 모바일 기기에 배포한다. 본인의 기기에서 게임이 실행되고 있는지 확인한다.
3. 기기와 컴퓨터를 연결하고 있는 USB 케이블의 연결을 해제한다.
4. 웹 브라우저를 연다. 크롬Chrome을 추천한다. 다음 URL을 주소창에 입력한다.

http://[Device IP Address]:55055/

5. 브라우저에 다음 스크린샷과 매우 비슷한 CUDLR 콘솔 창이 보일 것이다.

CUDLR 서비스 콘솔을 보여주는 브라우저

6. 타일 맵을 불러오면서 생긴 아홉 개의 요청 콜과 아홉 개의 응답 콜이 기록돼 있다. 이제는 이 요청들을 읽고 어떤 의미인지 이해할 수 있을 것이다.
7. 콘솔이 할 수 있는 것들은 또 무엇이 있을까? 콘솔 아래의 텍스트 박스에 help를 입력해보자. 실행 가능한 명령들의 리스트가 나올 것이다. 지금은 몇 개의 명령밖에 없지만 추후에 좀 더 추가할 예정이다. 만일 콘솔이 반응하지 않는다면 페이지를 새로 고치거나, 모바일 기기에서 게임이 실행되고 있는지 확인한다.

만일 CUDLR을 실행하거나 연결하는 데 문제가 생긴다면 10장, '문제 해결'을 읽어보길 바란다. 이전에도 언급했듯이 유니티로 개발하면서 디버깅과 문제를 진단할 수 있는 다른 방법도 알아볼 것이다. 하지만 온전히 원격으로 실행할 수 있는 CUDLR이 실제 움직임과 GPS 추적을 게임에서 테스트해볼 수 있는 가장 좋은 방법이다. GPS 이야기가 나왔으니 이제 이 장의 마지막 절로 들어가서 모든 부분을 하나로 모아보자.

GPS 서비스 설정하기

현실 위치 기반 지도를 생성하기 위해 필요한 마지막 조각은 기기의 위치가 어디인지 알아내는 것이다. 물론 가장 좋은 방법은 이전에 언급했듯이 기기에 내장돼 있는 GPS를 사용해 기기가 있는 위도와 경도 좌표를 얻는 것이다. 지도 타일에서 했던 방식으로 스크립

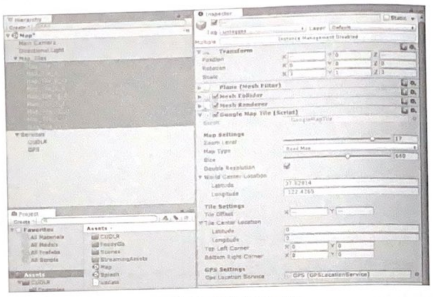
트를 가져와 서비스를 구성함으로써 추가적인 스크립트 제작 없이 단시간에 실행 가능하게 만들어보자.

시작하기 전에 다음 사항을 체크해서 본인 기기의 위치 서비스가 활성화돼 있는지 확인하자.

- 안드로이드: Settings > Location으로 가서 서비스가 켜져 있는지 확인한다.
- iOS: 다음 설명을 따른다.
  1. Privacy > Location Services를 탭한다.
  2. 아래로 스크롤하고 FoodyGO를 탭한다.
  3. 위치 접근을 Never로 할지 While Using the App으로 할지 결정한다.

이제 다음 설명에 따라 GPS 서비스 코드를 설치하고 게임을 테스트한다.

1. Hierarchy 창의 Services 아래에 있는 GPS 게임 오브젝트를 선택한다.
2. Inspector 창의 Add Component 버튼을 클릭하고, 컴포넌트 리스트에서 Services > GPSLocationService를 선택한다.
3. GPS 서비스 오브젝트에 GPSLocationService 컴포넌트가 추가된 것을 볼 수 있다.
4. Hierarchy 창에서 Map_Tile_0_0을 선택하고, 가장 아래에 있는 지도 타일을 Shift + 클릭(맥에서도 같음)해서 아홉 개의 지도 타일을 모두 선택한다.
5. 아홉 개의 지도 타일이 선택된 상태에서 Hierarchy 창에 있는 GPS 서비스 오브젝트를 끌어 Gps Location Service 필드에 놓는다.



동시에 여러 지도 타일을 수정하는 모습

6. 방금 한 작업은 아홉 개의 모든 지도 타일에 GPSLocationService를 동시에 추가한 것이다. `GoogleMapTile` 스크립트를 기억한다면 지도 타일들이 `GPSLocationService`를 호출해서 지도의 중심점 좌표를 구하는 사실을 알고 있을 것이다.
7. 이제 모든 지도 타일에 GPS 서비스가 연결됐으니 Play를 눌러 어떤 모습인지 알아보자.
8. 뭐가 잘못됐는지 몰라 혼란스럽더라도 걱정하지 말자. 아무것도 잘못되지 않았다. 컴퓨터상에서 실행되는 유니티 에디터는 실행 중 위치 서비스나 GPS에 접속할 방법이 없다. 이제는 기기에 게임을 배포해보자.
9. 빌드하고 배포한 후 본인의 기기에서 게임을 실행한다. 지금쯤이면 배포 과정이 자연스럽게 느껴질 것이다.
10. 이제는 주위를 보여주는 확대 지도가 보일 것이다. 지도가 조금 어긋나 보일 텐데, 카메라의 위치 때문이다. 다음 장에서 수정할 예정이니 걱정하지 말자. 혹시 지도를 보는 데 다른 문제가 있다면 위 단계들을 다시 한 번 확인해보자. 이전 단계들을 다 확인한 후에도 문제가 생긴다면 10장, '문제 해결'을 읽어보자.

11. 브라우저로 돌아가서 CUDLR 콘솔 페이지를 새로 고친다. 콘솔에 출력되는 것들을 둘러본다. 지도 타일 요청들을 주시해보자. 본인의 위치 좌표와 콘솔의 중앙점 좌표가 매치해야 한다.
12. 컴퓨터에서 기기를 뽑고 집 근처를 돌아다녀보자. 네트워크와의 연결이 끊어질 정도로 멀리 가지는 말고, GPS가 위치를 업데이트할 만큼만 안 돌아다니면 충분하다. 친구가 기기를 들고 돌아다니면서 본인은 CUDLR 콘솔을 보고 있는 것도 좋은 방법이다.

좀 더 가지고 놀면서 즐겨라.

GIS, 매핑, GPS에 대한 소개로 시작해 마지막 절까지 오면서 손에 든 결과가 만족스럽게 느껴졌으면 좋겠다.

요약

이 장에서는 GIS, 매핑, GPS에 대한 핵심 개념을 소개했다. 이 지식을 기반으로 Google Maps API를 유니티에서 불러와 다루는 데 필요한 용어들을 정의했다. 그런 후 게임에 지도를 추가했지만 품질이 떨어져 보였으며, 문제 해결을 위해 게임 지도를 타일화된 지도 시스템으로 구성했다. 구성이 끝난 후 잠시 주요 주제에서 벗어나 CUDLR이라 불리는 콘솔 디버깅 툴을 소개했다. CUDLR은 게임 핵심 부분의 디버깅을 가능하게 해주고, GPS를 통해 플레이어의 위치를 알려준다. 이 장의 마지막에서는 게임에 GPS 위치 서비스를 구성한 후 게임에 GPS를 추가했다.

핵심 부분들을 짚고 넘어갔으니 이제는 실제 게임 개발에 좀 더 집중할 수 있게 됐다. 다음 장에서는 신에 온전히 리그ᵗᵒᵍ된 캐릭터의 추가, 모바일 터치 입력, 자유 시점 카메라, 기기의 모션 센서 접근 등을 다룰 것이다.